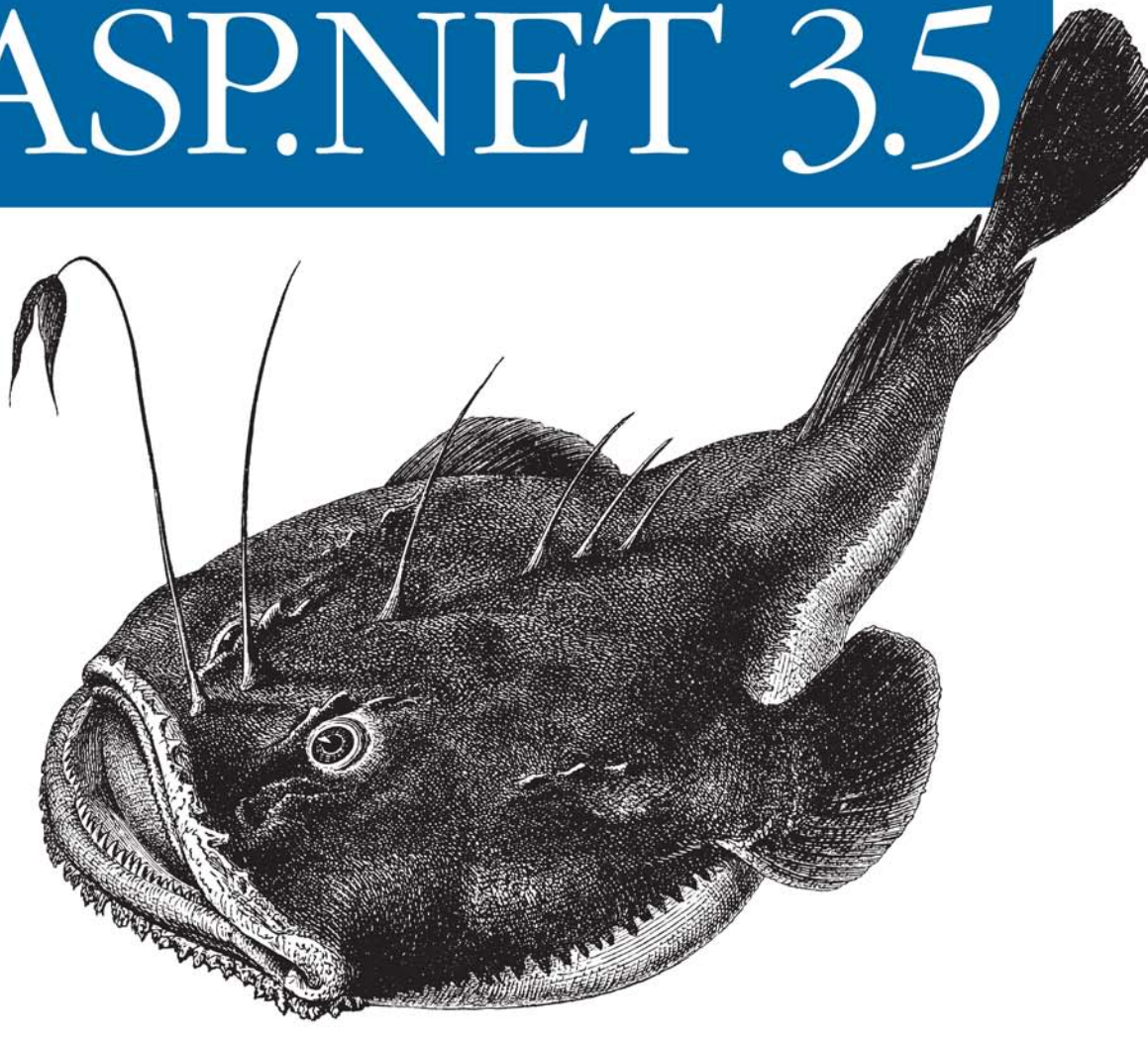
VISIT…

Build Web Applications with ASP.NET 3.5, AJAX, LINQ, and More

2nd Edition
New for Visual Studio 2008

Learning

ASP.NET 3.5



Jesse Liberty, Dan Hurwitz & Brian MacDonald

O'REILLY®

O'REILLY®

Learning ASP.NET 3.5

Learning ASP.NET 3.5, Second Edition, teaches you how to create engaging and interactive applications using the latest version of ASP.NET with AJAX and the productivity-enhancing features of Visual Studio 2008. All you need to get started is a basic knowledge of HTML and a desire to produce professional-quality websites.

Each chapter in this book is a self-contained lesson that introduces new skills—with illustrations and plenty of annotated examples—that you can use right away. You'll also get unique Brain Builders in each chapter that include practical exercises and review quizzes so that you can practice what you've learned and test your understanding. Inside, you'll find:

- A series of tutorials on different aspects of ASP.NET web development
- AJAX-integrated examples
- Examples that illustrate how new concepts work—each chapter includes one example with several stages, or a series of smaller examples
- VB, JavaScript, and SQL Cheat Sheet sidebars to help readers with little or no background on these topics
- An example web application in the final chapter that incorporates everything you've learned

If you want to get up to speed with the world's most popular web development technology, *Learning ASP.NET 3.5* is the best resource for the job.

"I'd recommend this book as a great introduction to the most commonly used aspects of ASP.NET. Readers who go all the way through will have touched on all the tasks that they're likely to encounter when they create (small) dynamic websites."

—Mike Pope, Microsoft User Education

Jesse Liberty is the bestselling author of O'Reilly's *Learning ASP.NET 2.0 with AJAX*, *Programming C#*, and *Programming .NET 3.5*.

Dan Hurwitz is the president of Sterling Solutions, Inc., where he provides contract programming and database development.

Brian MacDonald is the coauthor of O'Reilly's *Learning C# 2005*, and is the editor of several programming and networking books.

www.oreilly.com

US $44.99 **CAN $44.99**

ISBN: 978-0-596-51845-5

9 780596 518455 54499

Safari Books Online

Free online edition for 45 days with purchase of this book. Details on last page.

SECOND EDITION

Learning ASP.NET 3.5

Jesse Liberty, Dan Hurwitz, and Brian MacDonald

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

Learning ASP.NET 3.5, Second Edition
by Jesse Liberty, Dan Hurwitz, and Brian MacDonald

Copyright © 2008 Jesse Liberty, Dan Hurwitz, and Brian MacDonald. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: John Osborn
Production Editor: Sumita Mukherji
Proofreader: Sumita Mukherji
Indexer: Angela Howard

Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Jessamyn Read

Printing History:

September 2007: First Edition.

July 2008: Second Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *Learning ASP.NET 3.5*, Second Edition, the image of a monkfish, and related trade dress are trademarks of O'Reilly Media, Inc.

.NET is a registered trademark of Microsoft Corporation.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

This book uses RepKover™, a durable and flexible lay-flat binding.

ISBN: 978-0-596-51845-5
[M]

Table of Contents

Preface . . . . . **xi**

1. Getting Started **1**
Hello World 2
Creating a New Web Site 2
Creating HelloWorld 6
Making the HelloWorld Web Site Interactive 8
What You Just Did 13
Summary 13
Brain Builder **15**
Quiz 15
Exercise 15

2. Building Web Applications **17**
Mastering Web Site Fundamentals 17
The Page 17
Controls 19
Code-Behind Files 22
Events and Postbacks 22
Synchronous and Asynchronous Postbacks 23
The Page Load event and synchronous postback 27
Adding asynchronous postbacks 29
Using Controls 35
Organizing the Properties Window 35
Finding properties with IntelliSense 35
Basic Controls 36
Creating Tables 39

Setting Properties 41
Selection Controls 43
Panels 45
List Selection Controls 45
Adding items with the Item editor 46
Adding items in Source view 47
More Selection Controls 50
Using Selections to Display Text 52
Images 56
Links 57
LinkButtons 58
Source Code 58
Summary 63
Brain Builder 65
Quiz 65
Exercises 65

3. Snappier Web Sites with AJAX 69
Take a Walk on the Client Side 69
ScriptManager and UpdatePanel 71
Controlling Browser History 76
Extending Controls with the Control Toolkit 83
TextBoxWaterMarkExtender 85
PopupControlExtender 89
CollapsiblePanelExtender 95
Source Code Listing 100
Summary 105
Brain Builder 106
Quiz 106
Exercises 106

4. Saving and Retrieving Data 111
Getting Data from a Database 112
Binding Data Controls 113
Create a Sample Web Page 115

Using a DataSource Control 115
Pay No Attention to That Man Behind the Curtain 121
GridView Control 123
Auto-Generated Code 125
Adding Insert, Update, and Delete Statements 128
Displaying and Updating the Data 133
Take It for a Spin 134
Modifying the Grid Based on Conditions 135
Selecting Data from the GridView 139
Passing Parameters to the SELECT Query 140
LINQ 146
Creating the Object Model 147
Using the Object Model 153
Editing Data in LINQ 157
ASP.NET Dynamic Data 161
Source Code Listings 166
Summary 172
Brain Builder **174**
Quiz 174
Exercises 174

5. Validation **179**
Validation Controls 180
The RequiredFieldValidator 182
The Summary Control 188
The Compare Validator 189
Checking the Input Type 193
Comparing to Another Control 193
Range Checking 195
Regular Expressions 196
Custom Validation 198
Summary 201
Brain Builder **202**
Quiz 202
Exercises 202

6. Style Sheets, Master Pages, and Navigation 207
Styles and Style Sheets 207
Cascading Styles 208
Inline Styles 208
Pros and cons 210
Document-Level Styles 210
Pros and cons 211
External Style Sheets 212
Master Pages 220
Creating a Master Page 221
Adding Content Pages 225
Using Nested Master Pages 228
Changing the Master Page at Runtime 232
Navigation 234
Buttons and HyperLinks 235
Menus and Bread Crumbs 240
Site Maps 243
Using Sitemaps 246
TreeView 246
Customizing the look and feel of the TreeView 247
Replacing the TreeView with a menu control 248
Accessing site map nodes programmatically 249
Bread Crumbs 252
Summary 255
Brain Builder 257
Quiz 257
Exercises 257

7. State and Life Cycle 261
Page Life Cycle 261
State 267
View State 269
Session State 279
Application State 285
Summary 286
Brain Builder 288
Quiz 288
Exercises 288

8. Errors, Exceptions, and Bugs, Oh My! 291
Creating the Sample Application 292
Tracing 295
Page-Level Tracing 296
Inserting into the Trace Log 298
Debugging 300
The Debug Toolbar 302
Breakpoints 303
Setting a breakpoint 303
Breakpoints window 303
Breakpoint properties 305
Breakpoint icons 307
Stepping Through Code 309
Examining Variables and Objects 310
Debug Windows 310
Immediate window 311
Locals window 312
Watch window 313
Call Stack window 313
Error Handling 313
Unhandled Errors 314
Application-Wide Error Pages 316
Page-Specific Error Pages 320
Summary 320
Brain Builder 323
Quiz 323
Exercises 323

9. Security 326
Forms-Based Security 326
Creating Users with the WAT 327
Managing Users Programmatically 332
Creating User Accounts 333
Creating a Welcome Page 336
Creating a Login Page 337
Roles 340
Restricting Access 342

Testing for Login Status 345
Testing for Role-Based Authentication Membership 345
Summary 349
Brain Builder 351
Quiz 351
Exercises 351

10. Personalization 353
Profiles 353
Simple Data Types 354
Complex Data Types 359
Anonymous Personalization 364
Adding an Anonymous Profile 365
Migrating Anonymous Data to an Actual User's Record 370
Themes and Skins 371
Create the Test Site 372
Organize Site Themes and Skins 374
Enable Themes and Skins 375
Specify Themes for Your Page 377
Using Named Skins 381
Summary 382
Brain Builder 384
Quiz 384
Exercises 384

11. Putting It All Together 388
Getting Started 388
Adding Styles 389
Using Master Pages 392
Setting Up Roles and Users 396
Logging In 398
Navigation 402
Products Page 403
Adding AJAX 414
Cart Page 414
Purchase Page 419
Confirm Page 426
Custom Error Pages 428

Summary 429
Source Code Listings 430
Cart Page 430
Confirm Page 433
Home Page 435
Login Page 435
Master Page 436
Products Page 438
Purchase Page 442
Web.config 447

A. Installing the Stuff You'll Need 453

B. Copying a Web Site 466

C. Publishing Your Web Site 478
Database Support 479
Your Domain Name 479
Picking a Hosting Site 481
Setting Up the Account 481
Uploading the Web Site Files 482
Uploading Data 482
Creating the script 485
Logging Into the hosting database and running the script 487
Updating the Connection Strings 492
Notes and Tips 493
Running Web Server Software 495
Security 495
Configuring the Firewall 496
Hosting a Web Site Under IIS 498
IIS 7 498
IIS 6 499
Setting Up the Domain Name 500

D. Answers to Quizzes and Exercises 503

Index 575

Preface

ASP.NET 3.5 is arguably the fastest, most efficient, most reliable, and best-supported way to create interactive web applications available today. Combined with the development tools available from Microsoft (both free and commercial), it is incredibly easy to create web sites that look great and perform well. Best of all, most of the "plumbing" (security, data access, layout, and so on) is taken care of for you by the .NET Framework.

About This Book

This book will teach you how to build professional quality, interactive, robust data-driven web applications using Visual Basic 2008.

ASP.NET is not difficult to learn. All of the concepts are straightforward, and the Visual Studio and Visual Web Developer environments simplify the process of building powerful web applications. The difficulty in ASP.NET is only that it is so complete and flexible that there are many pieces that must be woven together to build a robust, scalable, and efficient application. This book cuts to the heart of the matter, showing in clear, easy-to-follow steps how to understand and build a web site.

ASP.NET makes it possible to create sophisticated and useful sites with minimal coding. You can enhance the functionality of your sites with Visual Basic or C#, if you choose, but the amount of code that you *must* write is surprisingly small. We've adopted that philosophy throughout this book, showing you how to take the most advantage of the tools Microsoft and ASP.NET provide while keeping the coding to a minimum. Whenever you need to write code, we walk you through each step and explain what it all does.

About This Series

O'Reilly Learning books are written and designed for anyone who wants to build new skills and who prefers a structured approach to studying. Each title in this series makes use of learning principles that we (with your help) have found to be best at equipping you with the knowledge you need for joining that new project, for coping with that unexpected assignment from your manager, or for learning a new language in a hurry.

To get the most out of any book in the Learning series, we recommend you work your way through each chapter in sequence. You'll find that you can get a quick grasp of a chapter's content by reading the instructional captions we've written for its examples and figures. You can also use the chapter Summary to preview its key takeaways and to review what you have learned. Most chapters feature one or more sample applications, and, if you learn best by reading code, you can turn to the complete source listing that appears just before the Summary. To bridge any gaps in your knowledge, check out the Cheat Sheets. Finally, to help you test your mastery of the material in each chapter, we conclude with a Brain Builder section, which includes a short quiz to test your grasp of the theory, and some hands-on exercises to give you practice building real applications with your new skills.

Learning books work with you as you learn—much as you would expect from a trusted colleague or instructor—and we strive to make your learning experience enjoyable. Tell us how we've done by sending us praise, brickbats, or suggestions for improvements to *learning@oreilly.com*.

Learning or Programming?

We have written two ASP.NET books: the one you are currently reading and another named *Programming ASP.NET 3.5* (O'Reilly). This book, *Learning ASP.NET 3.5*, is intended for beginning ASP.NET developers, and answers the question, "What is the quickest way for me to build real web applications with the least amount of coding?"

Our other book, *Programming ASP.NET,* is for developers who are saying: "Help me learn in depth—show me how everything works, and then help me put it to work in web applications." The key difference is this book is aimed to make you productive quickly, while the second book is designed to explore the technology in more depth. They complement each other, but if you are starting out and want to get to work fast, *this* is the one for you.

Learning ASP.NET 3.5 assumes you know some HTML and have some familiarity with Visual Basic 2008 (VB) or C#, or can pick up what you need along the way (or you're willing to run right out and buy *Programming Visual Basic 2008* by Tim Patrick [O'Reilly], although for what you'll be doing here, you won't really need it). To help with this, we have included VB Cheat Sheets throughout the book to explain and clarify some of the VB topics for newbies.

VB Versus C#

A quick note on Visual Basic versus C#: some people choose a .NET book based on what language the examples are given in. That's a natural reaction, but it's really not necessary, and here's why: there is very little actual VB or C# code in any given ASP.NET application, and what there is, you can easily translate from one to the other "on inspection." Besides, the two languages are strikingly similar, and both produce the same output. If you know one, it's quite simple to learn the other. In fact, there are software tools that can convert one language to the other with amazing accuracy. Finally, ASP.NET programmers benefit terrifically by being "bilingual"—that is, having the ability to read VB and write C# (or vice versa).

In the end, we had to choose one language over the other, and we elected to do the examples and exercises for this book in Visual Basic. However, if you prefer C#, you'll find every single example and exercise solution reproduced in C# free for download from this book's web site at *http://www.oreilly.com/catalog/9780596518455*.

How This Book Is Organized

Chapter 1, *Getting Started*, walks you through creating your first web site, *HelloWorld*.

Chapter 2, *Building Web Applications*, goes over the fundamentals of web sites and covers the basic controls available to you in ASP.NET.

Chapter 3, *Snappier Web Sites with AJAX*, shows you how to integrate this powerful client-side technology into your ASP.NET pages.

Chapter 4, *Saving and Retrieving Data*, shows you how to make your site interact with data stored in a database. You'll see controls to retrieve data, allow your users to interact with that data, and then save it back to the database. We'll also discuss the Language Integrated Query (LINQ), new to the .NET Framework.

Chapter 5 looks at *Validation*. ASP.NET provides extensive support for data validation, including ensuring that users provide required information, checking that values are within a range, and matching regular expressions.

Chapter 6, *Style Sheets, Master Pages, and Navigation*, shows you how to make web sites that are professional quality, good looking, consistent, and easy to navigate.

Chapter 7 examines *State and Life Cycle* in ASP.NET. Understanding how, and in what order, a page and its controls are created on the server and rendered to the browser is crucial for building successful interactive web sites. State is the current value of everything associated with the page. This is mostly handled automatically, but this chapter shows you how useful it can be to the developer.

Chapter 8, *Errors, Exceptions, and Bugs, Oh My!*, shows you how to use Visual Studio's tools to debug your application, and also how to handle errors in your code before users see them.

Chapter 9, *Security*, shows you how you can protect your web site from malicious users. You'll find out how to register your users and how to hide parts of your site from users who don't have the appropriate privileges.

Chapter 10, *Personalization*, shows you how to allow your end users to customize the look and feel of the web site according to their personal preferences. You will see how to use themes and skins to accomplish this.

Chapter 11, *Putting It All Together*, is a single, large example that integrates almost everything you have learned throughout the book.

Appendix A, *Installing the Stuff You'll Need*, tells you what hardware and software is required to run the examples in this book and helps you set up your environment.

Appendix B, *Copying a Web Site*, describes the process of copying a web site to a new web site. This is a technique used often throughout this book when building up examples.

Appendix C, *Publishing Your Web Site*, covers the steps to take a web site from the development stage to being publicly available on the web.

Appendix D, *Answers to Quizzes and Exercises*, presents detailed solutions to all of the quiz questions and practice exercises found at the end of each chapter.

Conventions Used in This Book

The following font conventions are used in this book:

Italic
: Used for pathnames, filenames, program names, Internet addresses, such as domain names and URLs, and new terms where they are defined.

`Constant width`
: Used for command lines and options that should be typed verbatim, and names and keywords in program examples. Also used for parameters, attributes, properties, expressions, statements, and values.

`Constant width italic`
: Used for replaceable items, such as variables or optional elements, within syntax lines or code.

`Constant width bold`
: Used for emphasis within program code examples.

Pay special attention to notes set apart from the text with the following icons:

This is a tip. It contains useful supplementary information about the topic at hand.

This is a warning. It helps you solve and avoid annoying problems.

Support: A Note from Jesse Liberty

I provide ongoing support for my books through my web site. You can obtain the source code for all of the examples in *Learning ASP.NET 3.5* at:

http://www.LibertyAssociates.com

There, you'll also find access to a book support discussion group that has a section set aside for questions about *Learning ASP.NET 3.5*. Before you post a question, however, please check my web site to see if there is a Frequently Asked Questions (FAQ) list or an errata file. If you check these files and still have a question, then please go ahead and post it to the discussion center. The most effective way to get help is to ask a precise question or to create a small program that illustrates your area of concern or confusion, and be sure to mention which edition of the book you have (this is the second edition).

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books *does* require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Learning ASP.NET 3.5*, by Jesse Liberty, Dan Hurwitz, and Brian MacDonald. Copyright 2008 Jesse Liberty, Dan Hurwitz, and Brian MacDonald, 978-0-596-51845-5."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *http://permissions@oreilly.com*.

We'd Like to Hear from You

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/9780596518455

To comment or ask technical questions about this book, send email to:

http://bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com/

Visit the O'Reilly .NET DevCenter:

http://www.oreillynet.com/dotnet

Safari® Books Online

When you see a Safari® Books Online icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://safari.oreilly.com*.

Acknowledgments

From Jesse Liberty

I am particularly grateful to John Osborn, who has shepherded all of my work through O'Reilly, as well as the editors and production folks at O'Reilly who (as always) made this book so much more than what we originally created.

From Dan Hurwitz

In addition to the people mentioned by Jesse, I also want to commend Brian for the great work he has done on this edition. It is a much better book because of his efforts. And, as always, I especially want to thank my wife for being so supportive of this project. It sounds trite and repetitious, but it would not be possible without her help.

From Brian MacDonald

As always, my deepest appreciation goes to Jesse and Dan for inviting me to be a part of this project, and for having me back for another edition. My thanks also to John Osborn for getting me involved with O'Reilly in the first place, many years ago now. My gratitude and appreciation to our technical reviewers, Jesudas Chinnathampi, Owen Davies, and especially to Mike Pope, who took no prisoners, but whose feedback improved the quality of this book tremendously. Sumita Mukerji, production editor extraordinaire, went above and beyond the call of duty several times, and I thank her for that. Many thanks to my wife, Carole, who once again provided technical and moral support, and to my son, Alex, for his patience while Dad worked on yet another chapter. Finally, thanks to my parents, Reenie and Dave, for fostering my love of reading and writing, and also for the gift of the lap desk that I used while writing this book.

CHAPTER 1
Getting Started

Learning ASP.NET 3.5 will teach you everything you need to know to build professional quality web applications using Microsoft's latest technology, including ASP.NET 3.5 and AJAX. ASP.NET is Microsoft's tool for creating dynamic, interactive web pages and applications. Using plain vanilla HTML, you can make a web page that has some great content, but it's *static*—the content doesn't change, no matter what the user does. You can even use Cascading Style Sheets (CSS) to make it the most visually impressive thing on the Web, but if what you really need is for users to be able to leave comments, or browse your inventory, or buy things from you, then HTML alone won't get it done.

That's where ASP.NET 3.5 comes in. Within these chapters, you'll find out how to do all the great tricks that you see on the most popular commercial web sites. Order forms? We've got that. Interact with a database? You'll do that, too. Dynamic navigation tools? It's in here. Personalized appearance that the user can customize? No problem.

The best part is, you'll do it all with minimal coding. You can make ASP.NET pages in your favorite text editor if you want, but that's a bit like using a hammer and chisel to write the Great American Novel. If you use Visual Studio 2008, or its free counterpart, Visual Web Developer, adding many features to your page is as simple as dragging and dropping. The tools generate most of the code for you. If you're an old-school type who cringes at the idea of letting someone else write your code, it's all still there, and you can tweak it to your heart's content. Consider this, though: would you rather spend your time writing the code for another radio button list, or figuring out what to do with the data that you gather using it? In short, the tools do the tedious chores for you, so you can get to the good stuff.

On top of all this, you can enhance your ASP.NET 3.5 site with AJAX, which is more than just résumé enhancement—it's a genuine improvement in the user experience. When a user is browsing your product catalog, and she clicks on one of your thumbnail images to view the product's details in another panel, she simply expects it to *work instantly*. She doesn't want to wait while the page contacts your server, reloads,

and then redraws itself with the new information. With AJAX, she won't see any of that. The update is seamless, and the user never has to slow down. You'll see AJAX tools used throughout this book. In fact, Chapter 3 is dedicated solely to just that topic, so you can use AJAX with everything else we'll show you.

One of the wonderful characteristics of the tools (Visual Web Developer or Visual Studio) and the technology you'll be using (ASP.NET and ASP.NET with AJAX) is that you'll be able to create your applications by dropping controls onto the page and just a little bit of handcoding to handle "events" (such as what happens when the user clicks a button). Not *toy* applications—meaningful business applications.

By the time you've finished this book, you'll be able to do all of that and more, and you'll learn about it by doing it yourself—hands-on. If you don't have Visual Studio or Visual Web Developer installed yet, turn to Appendix A now for detailed instructions on how to install and set it up. Once you've done that, it's time to dive right in and create your first application, "Hello World."

Hello World

One of the most difficult problems in beginning to learn any programming technology is the "bootstrap" problem. That is, writing your first program requires using techniques that you haven't learned yet, but learning those techniques in a vacuum is not only boring, but to some degree pointless because there's no context, and thus no way to integrate that which you learned.

The traditional solution to this dilemma is to create the canonical "Hello World" program. Our Hello World web site will allow us to demonstrate many useful aspects of ASP.NET without overwhelming you with detail. We promise we will explain every aspect of this web site in detail as we go along.

According to Wikipedia (*http://en.wikipedia.org/wiki/Hello_World*), the tradition of a Hello World program dates back to a 1974 Bell Laboratories memorandum by Brian Kernighan.

This introductory web site will have only a Button and a Label control. Initially, the Label will display the text "Label." When the user clicks the Button, the Label text becomes "Hello World." Very cool, eh? You can see the finished product in Figure 1-1 as it appears after you've clicked the button.

Creating a New Web Site

To get started, open the Integrated Development Environment (IDE), which for your purposes in this book is Visual Web Developer or Visual Studio. (Throughout this book, we will use the acronym IDE for both, specifically using Visual Studio or Visual Web Developer only where they are different.)

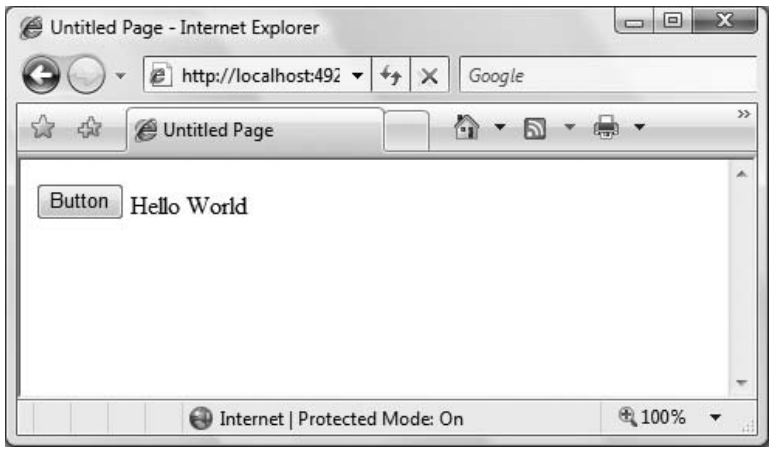


Figure 1-1. *This is what the HelloWorld web site will look like after you click the Button. It may not look like a lot, but this is a fully functional, interactive web site.*

To create a new web site, click on the menu item File → New Web Site…, or alternatively, use the Create: Web Site… link on the Start Page. Either way, you should see the New Web Site dialog, like the one shown in Figure 1-2.

In this book, we will be using Visual Basic as our default language, although it is our profound belief that Visual Basic and C# are really a single language, just with slightly different syntax.

We will be showing many of our screen shots from Visual Web Developer, because it is freely available from Microsoft; however, anything that you can do in Visual Web developer can also be done in Visual Studio.

Take another look at Figure 1-2, and we'll examine it in some detail. In the upper part of the window, you are offered various Visual Studio templates (though yours may vary, depending on any features and add-ons you've installed). Select the ASP.NET Web Site template because that is the kind of site that you are going to create (shown highlighted in this figure).

In the Location drop-down box at the bottom of the dialog box, select *File System* (the other options are HTTP or FTP; we'll explain this selection in the next section).

The Location drop-down in Figure 1-2 covers up another drop-down in which we have set the language to Visual Basic (rather than to Visual C#). Finally, you need to specify where on your disk you would like this web site to be placed—in this case, in the *LearningASP* directory on the C drive. You won't have such a directory on your hard drive yet, so you'll need to type it into the Location field (or whatever directory you want to use for your projects in this book).

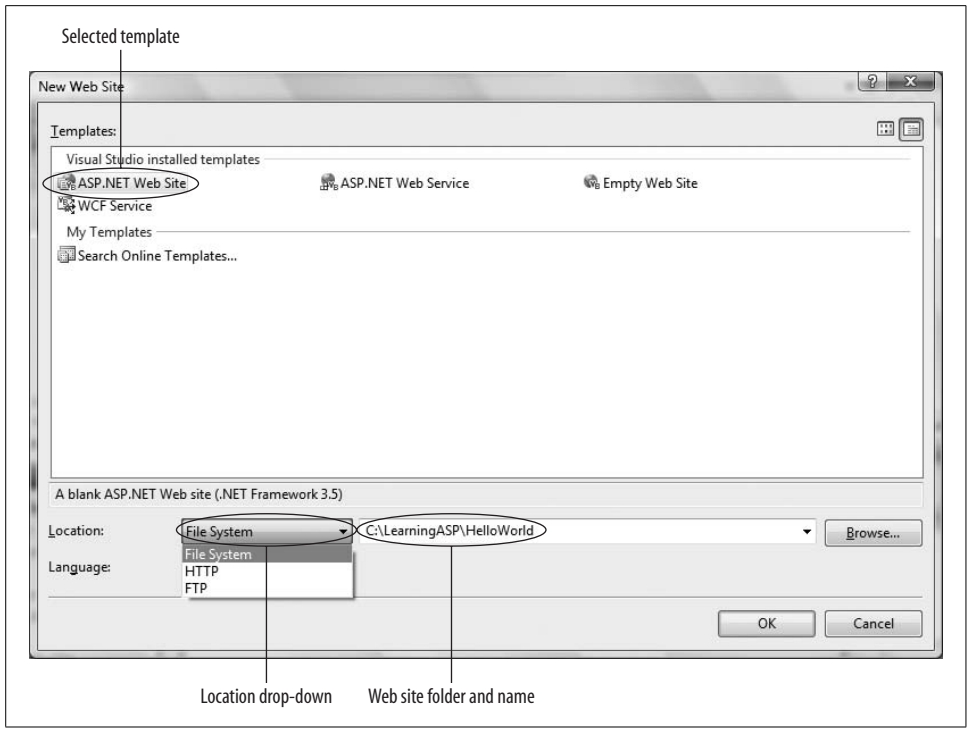


Figure 1-2. *To create a new web site, open the IDE, and click on Menu → New Web Site to open the New Web Site dialog box. The Visual Studio Installed templates and My Templates panels show you the types of sites supported by your version of Visual Studio.*

The name of the new web site will be *HelloWorld* (with no space character), so type that into the Location field. The site will be fully contained in a subdirectory named *HelloWorld* within the directory *LearningASP*.

Click OK; the IDE will create the directory for you, put the needed default files for your site within it, and then open the new web site.

You can confirm that the files are in the right place by navigating to the specified directory using Windows Explorer, as shown in Figure 1-3. When you work on your site, however, you'll most likely access these files through the Solution Explorer window located on the right side of the IDE window.

The Location field in Figure 1-2 is really composed of two parts: a drop-down with three possible values, and a text box for the folder name and path. The drop-down choices are File System, HTTP, and FTP.

File System is the default choice for new web sites and the only choice we'll be using in this book. It creates a new web site folder somewhere on the physical file system, either on your local machine or your network. One important feature of ASP.NET is that an entire web site can be contained within a directory. This is convenient not

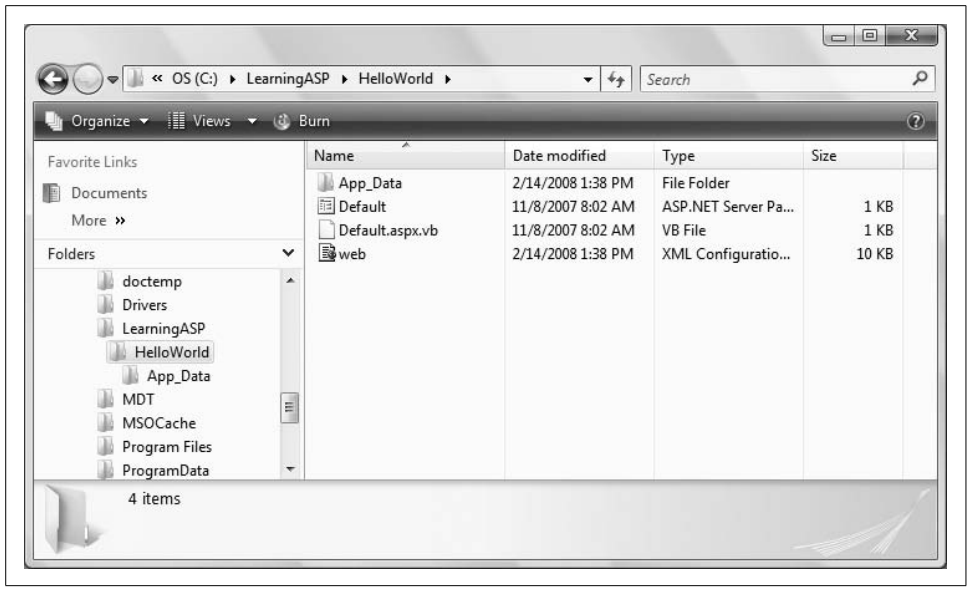


Figure 1-3. *Visual Studio creates a new web site directory for you, complete with some default files to start with.*

only for deploying your web site to a web server, but as a side benefit, it allows us to easily place samples from this book onto our web site for you to download and try on your local machine.

When you create your web site, you can use the Browse button (lower-right corner in Figure 1-2) and its associated drop-down list to browse the file system as you would with Windows Explorer; you can also select any desired folder as the "home" for your new web site folder.

When you run your file system-based web application from within the IDE, the development environment runs the application using its own internal web server rather than a web server program such as Microsoft Internet Information Server (IIS). This means that you can easily develop web sites on your machine without the necessity of installing IIS.

The alternatives to hosting your site in your file system are named HTTP and FTP. HTTP indicates that IIS (the web server product from Microsoft) will be serving the pages and requires that the web application be located in an IIS virtual directory. If you choose HTTP, the IDE will automatically create this virtual directory for you and the web site will be served by IIS.

FTP allows you to develop your web site on a remote location accessible via the FTP protocol. You will be presented with an FTP Log On dialog box with a checkbox to allow Anonymous Log in, and text boxes for login user name and password, if necessary.

Creating HelloWorld

After you've named your new web application and chosen a place to keep the files, the IDE will look more or less like Figure 1-4. This is where you do the real work of putting your site together.

Which exact windows you see and how they are presented may be determined by the options you've chosen. In your IDE, you can always open new windows from the View or Window menu, and you can undock, move, and redock the various windows using the mouse and the on-screen docking indicators.

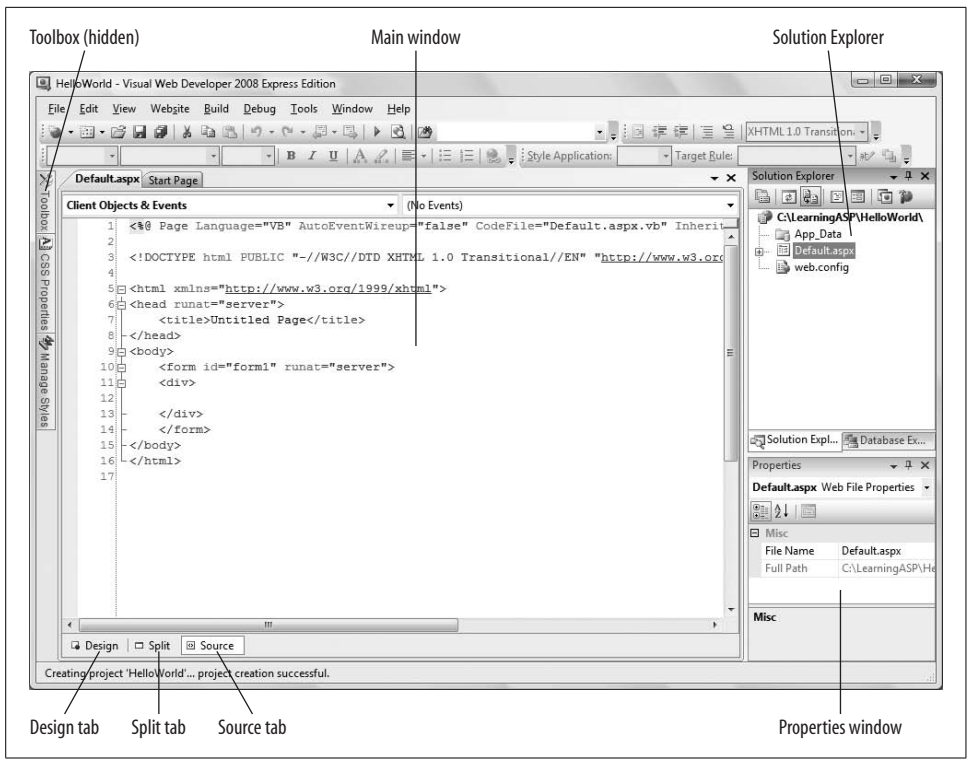


Figure 1-4. *Initial IDE screen for HelloWorld. This is what you'll see after you've named your web site, chosen a language, and created a directory for it.*

In Figure 1-4, you see the main window, which shows the page markup: HTML plus ASP.NET declarations and controls. This is called the markup file, or the *.aspx* file, because *.aspx* is the file extension associated with markup files. Also note the three tabs at the bottom of this pane, labeled Design, Split, and Source. You'll be using these three tabs a lot as you create your pages.

To start, click on the Design tab. When you click this tab, the middle window of your IDE becomes the *design surface*. On the design surface, you can drag and drop items such as buttons, text fields, labels, and so on from the Toolbox (which you'll see in a moment), where they automatically become part of your application. Each item that you can drag onto the design surface is called a *control*. You'll be reading more about controls in Chapter 2 and throughout this book.

Next, click on the Source tab. This view allows you to see the same controls, but displayed as HTML and ASP.NET markup. When you drag a control onto the design surface, the IDE automatically adds the appropriate markup to make that control part of the page. You can view and adjust that markup from the Source tab and even drag controls from the Toolbox directly onto the Source view. As you switch back and forth between the Source and Design views, they will remain consistent with one another, as they are two views of the same information.

Many working programmers—and even Microsoft itself—will refer to markup as source code. Other programmers draw a distinction between markup (HTML, ASP.NET controls, XML, etc.) on the one hand, and source code (C#, VB.NET, JavaScript) on the other. This can—and does—cause confusion, and all ASP.NET programmers learn to differentiate as best we can by context. The Source tab shows markup or HTML source code. The "code-behind" file, discussed below, shows C# or VB.NET source code. Not a perfect naming system, but there you have it. In practice, markup and ASP.NET source code have become synonymous.

Finally, click on the Split tab—here you'll see the best of both worlds. The Split view combines shows the Source view in the top pane, and the Design view in the bottom pane. You can drag controls onto either pane and the other pane will adjust appropriately. Sometimes it might take a moment or two for the other pane to catch up and synchronize, but you'll see a warning message when they're out of sync.

Again, referring to Figure 1-4, the window at the right edge of the screen displays the Solution Explorer, which is used for navigating and working with the files that make up your web site. The Database Explorer tab (called the Server Explorer in Visual Studio) at the bottom of the Solution Explorer window allows you to create and work with data connections.

Below the Solution Explorer window is the Properties window, which displays the *properties* for the currently selected object on the page. Properties are settings that are specific to each control, so the content of this window changes depending on what control you've clicked on. You'll be reading a lot more about properties in the discussion on controls in Chapter 2.

On the left edge of the Main window, click on the Toolbox tab to display the Toolbox. By default, the Toolbox will obscure part of your code window, so click the pushpin button in the title bar of the Toolbox to "pin" it in place and shove the code

window over a bit. You can "pin" any of the auxiliary windows in place, keeping them visible. When "unpinned," they will auto-hide, showing only their tab. Clicking on a tab while unpinned will make them temporarily visible.

Inside the Toolbox, you'll find a number of expandable categories that contain just about every control you'd want to use on your web page. If the Toolbox tab is not visible, click on View → Toolbox to display it. Initially it will be displayed in expanded view, as shown on the left side of Figure 1-5. Click on the + or – icon to collapse or expand each section.

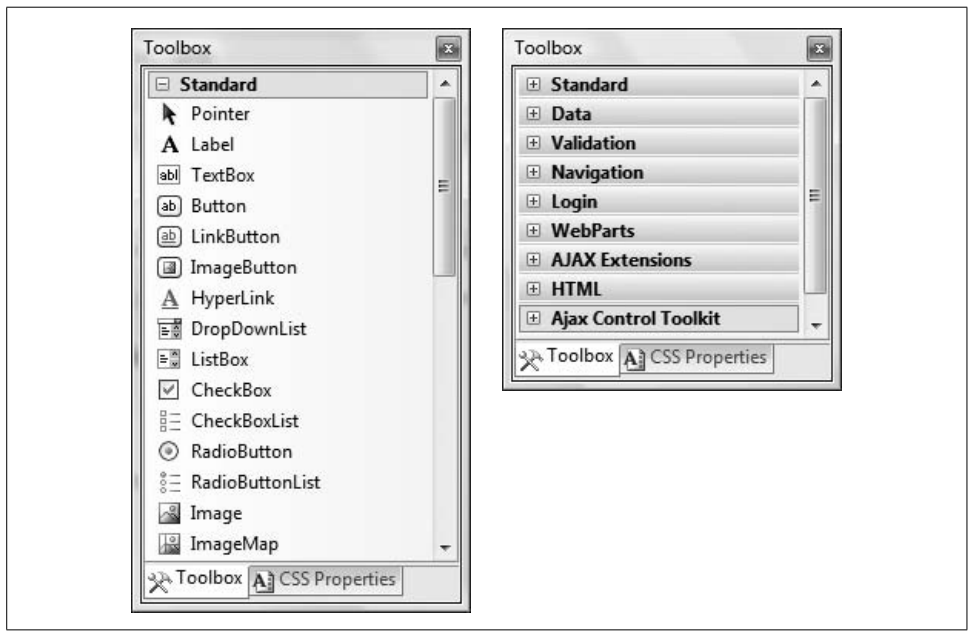


Figure 1-5. *The Toolbox provides quick access to just about everything you'd want to put on your page. Here, the Toolbox is shown expanded on the left and collapsed on the right.*

Making the HelloWorld Web Site Interactive

Although it doesn't seem like you've done much, you've already created your web page. It just doesn't do much of anything right now. To make your page come alive, you need to add some *controls* to it. Everything that you'll find in the Toolbox is a control, and you can add controls to your pages simply by dragging them onto the design surface or into the Source view.

For this first program, you'll add a button and a label to your page, making it look like what you saw back in Figure 1-1. Follow these steps:

1. Click the Design tab at the bottom of the main window to ensure that you are in Design view.

2. If the Toolbox window is not already pinned in place, click on the pushpin icon in its title bar to pin it in place.
3. If the Standard category of the Toolbox is not expanded, click on the plus symbol and expand it. You should be able to see a number of basic controls listed in the Toolbox, such as "Label," "TextBox," and "Button."
4. Click on a `Button` control in the Toolbox and drag it onto the design surface inside the `div` element.
5. Click on a `Label` control in the Toolbox, and drag that onto the design surface next to the button.

At this point, your IDE should appear similar to Figure 1-6.

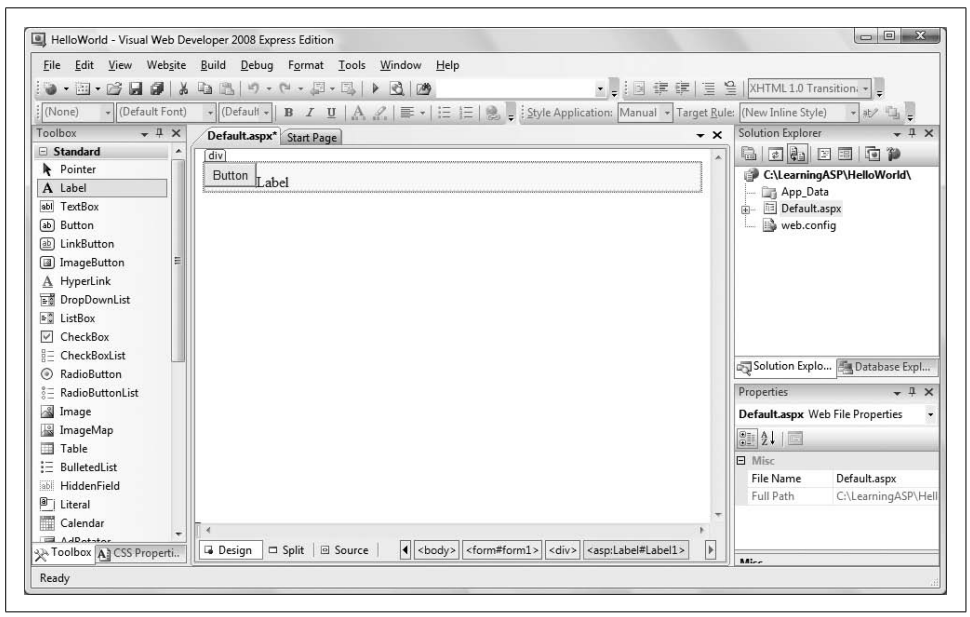


Figure 1-6. *After you've added the button and label to your HelloWorld application, the Design view should look like this.*

This is a good time to stop and run your program to see what it does so far. There are three ways to do this:

- Click on the menu item Debug → Start Debugging
- Press the F5 keyboard shortcut
- Click on the Start Debugging icon (▶) on the toolbar

Because this is the first time you've run the program, the IDE will detect that your application (specifically, its *web.config* file) is not set to allow debugging and will offer to make that adjustment for you, as shown in Figure 1-7. Get used to this dialog box—you'll see it the first time you run any web site.

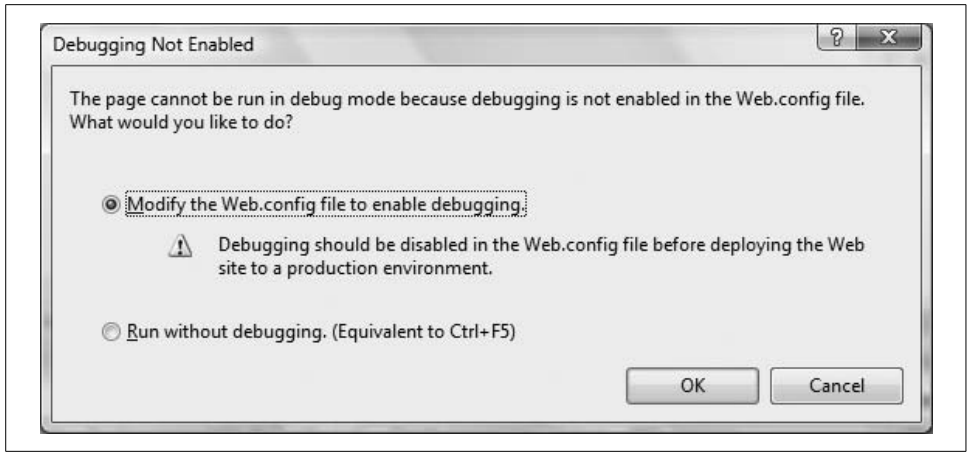

Debugging Not Enabled
The page cannot be run in debug mode because debugging is not enabled in the Web.config file.
What would you like to do?
Modify the Web.config file to enable debugging.
Debugging should be disabled in the Web.config file before deploying the Web site to a production environment.
Run without debugging. (Equivalent to Ctrl+F5)
OK
Cancel

Figure 1-7. *You'll see this Debugging Not Enabled dialog box the first time you run your application. Just select the first option and click OK to keep going.*

It's not important to know what a *web.config* file is right now; we'll explain it later. For now, click OK to allow the IDE to modify the configuration file.

After you click OK, your application begins, your default browser opens, and the page that contains your button is displayed, as shown in Figure 1-8.

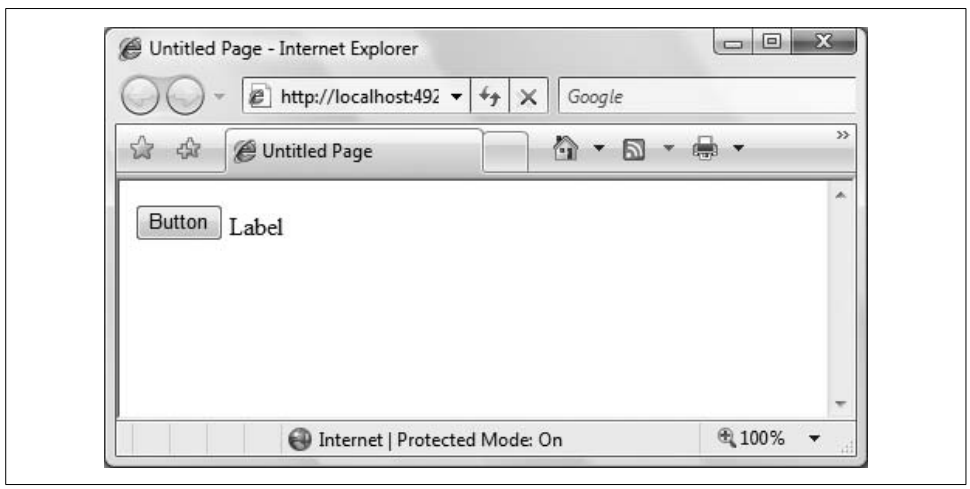


Figure 1-8. *This is what HelloWorld looks like in the browser before you do any coding. The button doesn't do anything yet, though.*

In the address bar of the browser, where you normally see the URL of the web site you're visiting, you'll see `http://localhost:`, followed by a number. This is still a URL, and "Localhost" is just what it sounds like—it represents the ASP.NET web server running on your local machine. The number is a port number, and will probably vary each time you run your application.

Click the button. Unfortunately, nothing happens. In the status bar of the browser, you may see evidence that the page is sent back to the server—this is called a *post back*, which we'll describe in Chapter 2. For now, close the browser to end the application, then return to the Design view in the IDE. You may see a new window named "Output" at the bottom of the IDE. If so, just close the window and don't worry about that for now.

All web applications are "event-driven." Any action that your control can take, or that a user can take with a control—clicking a button, opening a tool tip, checking a checkbox—is an *event*. Put simply, a web page without any events just sits there. When an event occurs, it is said to be *raised,* (we might also say that the event is *fired*). Then, behind the scenes, a block of code called an *event handler* is *called* to respond to that event. All of this firing of events and calling event handlers is automatically wired in by ASP.NET, and is far easier to set up than it is to describe, so *don't panic!*

Not surprisingly, all buttons have a default event named `Click`. The `Click` event is automatically fired whenever the user clicks a button. At the moment, when you click the button on your web page, the event is fired, but there's no event handler yet, so nothing happens.

Creating the event handler for the `Click` event is easy. In Design view, all you have to do is double-click the Button control. This instructs the IDE to create an event handler and name it. The IDE will name your event handler as follows: the ID of the control, followed by an underscore, followed by the name of the event. If you do not explicitly provide the ID for a button (we'll discuss naming events and event handlers later), the IDE will apply the default ID of `Button1`. Thus, the name of the event handler will be set to `Button1_Click`.

The IDE then displays the *code-behind* file and an event handler stub for you to complete. Here, you can add your own custom code to specify the operations you want the handler to perform when it's called.

Give it a try. Switch to Design view and double-click on the button. The code-behind file containing your newly created event handler will open, as shown in Figure 1-9.

Don't worry about any of the other code you see, like the `Partial Class` and `End Class` lines; that's all added automatically by Visual Studio. All you need to know right now is that your event handler is the code that starts with `Protected Sub` and ends with `End Sub`. You'll add your code in between those lines.

In this event handler, whenever the user clicks the button, you want to set the `Text` property of the `Label` control, which the IDE named `Label1`, to the phrase "Hello World." To do that, you need to assign that string of letters to the `Text` property of the Label. The event handler for `Button1_Click` appears as shown in Example 1-1.

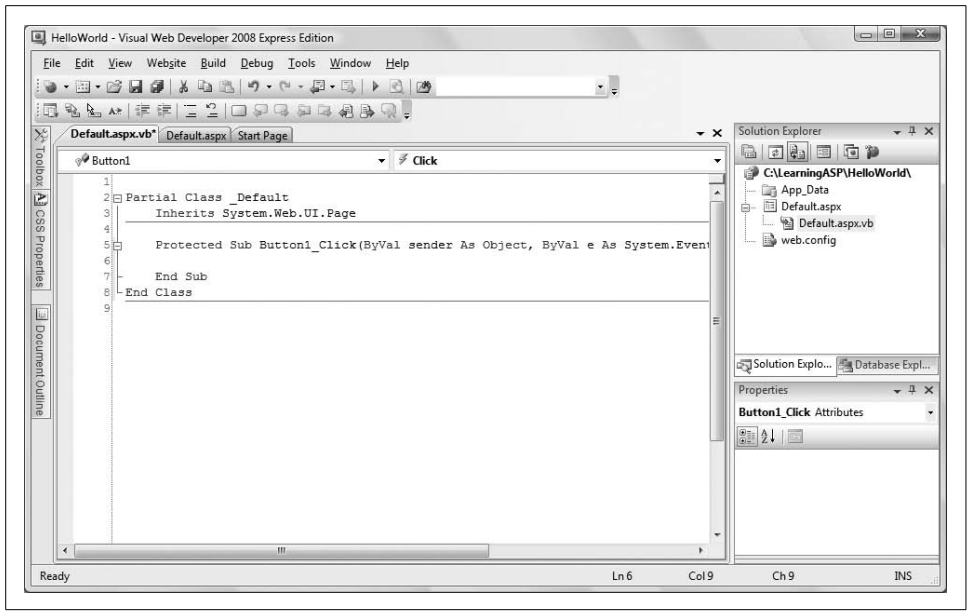


Figure 1-9. *When you double-click the button in HelloWorld, you'll be taken to the code-behind page, where you can modify the event handler yourself.*

Example 1-1. *The Button1_Click event handler in HelloWorld, before you change it*

```
Protected Sub Button1_Click( _
       ByVal sender As Object, _
       ByVal e As System.EventArgs) _
       Handles Button1.Click
End Sub
```

In this listing, and in other listings throughout this book, we've reformatted the code to fit the width of the printed page. In Visual Basic, the line continuation character is the underscore (as seen at the end of the first three lines, used here and elsewhere to make valid VB code). You, or the IDE, may place much of that code on a single line without the continuation character(s).

To assign the text string to the label, click inside the event handler, and then type the following code:

```
    Label1.Text = "Hello World"
```

When you're done, the event handler should look like Example 1-2.

Example 1-2. *The Button1_Click event handler, after your addition*

```
Protected Sub Button1_Click( _
       ByVal sender As Object, _
       ByVal e As System.EventArgs) _
       Handles Button1.Click
            Label1.Text = "Hello World"
End Sub
```

After you've added the code to your event handler, save the file by clicking the Save button on the toolbar, then run the program again by clicking the Debugging icon or pressing F5. When the page opens in your browser, click the button. Your event handler is working now, so you should see the text label change to "Hello World," as displayed back in Figure 1-1.

What has happened is that when you click the button, the page is sent back to the server, where your event handler code runs, and the string "Hello World" is assigned to the `Text` property of the `Label` control. A new page was created by the server and sent back down the "wire" to the browser as pure HTML, and then displayed on your screen. Close your browser to stop the application and return to the IDE.

What You Just Did

When you follow step-by-step instructions as if following a recipe, it's easy to lose sight of what you've done. Here's a quick review:

- You created a new web site on your file system.
- You dragged a `Button` and a `Label` onto the design surface.
- You double-clicked on the `Button` to create an event handler.
- In the event handler, you assigned "Hello World" to the `Text` property of the `Label` control.
- You ran your application and clicked on the `Button`, causing the page to be sent back to the server where the event handler code ran. The text "Hello World" was assigned to the `Label` and the page was sent back to the browser.

Congratulations! You've just built your first bona fide web page—and it's interactive, too. Pretty easy, isn't it? You've seen how to use the IDE, you've worked in Design view and in the code-behind file, and, most important, you saw how to create a page that actually responds to user input.

Summary

- ASP.NET 3.5 lets you create interactive web pages and applications. With dynamic pages, you can interact with your users and create a richer experience for them.
- Visual Studio 2008, or the free Visual Web Developer, supplies the tools that make creating a web page as easy as dragging and dropping, minimizing the code you need to write.
- AJAX is a set of technologies that you can use to make the user's experience more responsive.

- You can create a new web site or open an existing one from the Start Page in Visual Web Developer or Visual Studio.
- In ASP.NET, you can store your entire web site within a single directory, which in this book will always be on your local hard drive, but you can also store them at a remote location and serve them using IIS.
- The main window of the IDE has three views: Design, Source, and Split. Design view allows you to see the visual design of your web page; Source view shows the HTML and ASP.NET markup instead; and Split shows both views in separate panes. You can switch between the views on the fly.
- The items that you add to your web page are called controls. Controls are stored in the Toolbox, which by default appears on the left side of the IDE. You add controls to the page simply by dragging them from the Toolbox onto the appropriate spot on the page in Design view or Source view.
- The Solution Explorer, located on the right side of the IDE, displays the files in your web site. Below the Solution Explorer is the Properties window, which lets you adjust the properties of any control you select. On a separate tab is the Database Explorer (called Server Explorer in Visual Studio) for access to the databases that support your web site.
- You can run your application by clicking Debug → Start Debugging from the menu, pressing F5, or clicking the Start Debugging button.
- Web applications are event-driven, meaning that the controls raise events, which are handled by code blocks called event handlers.
- The code for the server controls resides in another file called the code-behind file.
- When you double-click on a control in Design view, you're automatically taken to the code-behind file, where the IDE will create a handler for the control's default event.

You've come a long way in just one chapter. Sure, "Hello World" is a trivial page, as web pages go, but it's interactive, which is the point of the book. You can close out the chapter with some quiz questions to see how much you've learned and then a simple exercise to let you practice your skills. Even though you've come this far, you've just scratched the surface of what's available to you in ASP.NET. Just glancing at the Toolbox shows you that there are many more controls than you've used in this chapter. In Chapter 2, we'll show you how to use some of them, and you'll build an order form to see how they all work together.

Quiz

1. How do you create a new web site in the IDE?
2. What are the three views of your page that you can use in the IDE?
3. What's the name for the settings that are specific to each control?
4. Where in the IDE will you find the controls that you can place on your page?
5. How do you run your application?
6. What event is fired when you click on the `Button` control?
7. Where is the code for the event handler located?
8. What's one way to access the default event handler's code?
9. What property of the `Label` control do you use to set its content?
10. When you click the button in your Hello World application, where is the code processed?

Exercise

Exercise 1-1. This is your first exercise, so we'll take it easy on you—you'll make some changes to *HelloWorld*. Open the example again. There are a few ways to do this:

- Select File → Open Web Site.
- Click the Start Page tab at the top of the main window to display the Start Page, and click the Open Web Site link, or select it from the Recent Projects list, if it's there (and it should be, if you've just finished this chapter).

With the file open, select the code-behind file, either from the tab at the top of the window, or in Solution Explorer. Go to the `Click` event handler, and change the "Hello World" text to a message of support for your favorite sports team (or band, or movie, or whatever you like).

Now switch back to the *.aspx* file. Select the `Label` control, and check out the Properties window. There's more here than just setting the text, as you've seen. Go to the Appearance section of the Properties, and play around with them to your liking. Click the + sign next to the `Font` property, and you'll find lots of options that you're probably familiar with. Try changing the font, the text size, and the color. You can also play with the border of the label, too. Note that if you change the `Text` property here, you're changing the initial text that the label displays. After you've kicked the tires a bit, run your application to see how it looks. You can see an example in Figure 1-10, although this is the affiliation of only one of the authors.

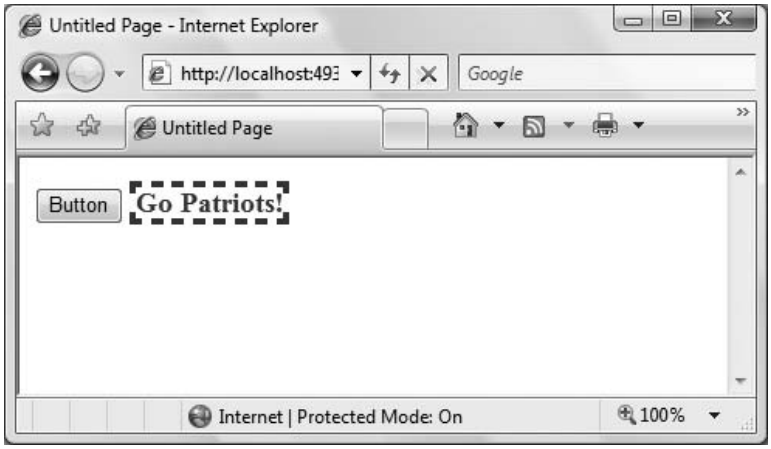


Figure 1-10. *The results of Exercise 1-1, for at least one of the authors. Your page may look different, depending on your sports loyalties.*

CHAPTER 2

Building Web Applications

You've built your first web site, and you've gotten your feet wet, which is great. But so far, you've only used two controls: `Label` and `Button`. You've seen the Toolbox in the IDE, and it's stuffed with controls just waiting for you to experiment with. That's exactly what you're going to do in this chapter. You'll build a functional order form for a fictional business, even though you won't do anything just yet with the data your form will collect. You'll get to try out many of the basic controls in Design view and Source view; you'll learn about web site fundamentals, selection controls, and their collections of items, and you'll see how to display the results retrieved by one control in another control somewhere else on the page.

Mastering Web Site Fundamentals

The difference between a web *page* that simply displays information and a web *application* that interacts with your user is the application's ability to gather data from the user, process it on the server, and take action accordingly. The core of a web application is the page and its interactive controls. This part of the chapter will introduce the web page and the types of controls that you'll use throughout the remainder of this book and throughout your ASP.NET programming career. We will also introduce the mindset that will move your applications from being a "brochure" that displays information into an interactive application delivered over the Web.

The Page

Every ASP.NET web site consists of at least one web page stored in a single file with the extension *.aspx*. There is usually more than one file, as you saw in Chapter 1. The *.aspx* file is called a *content* file. Some developers call it the *markup* file, which makes sense when you remember that HTML stands for HyperText Markup Language.

The contents of the page itself are composed of "server controls" and "normal" HTML. Server controls are simply controls with code that runs on the server. Normal HTML in the *.aspx* file is sent to the browser "as is." All you really need to know to create ASP.NET pages is that HTML is rendered at the browser, and the controls are processed on the server. If you want to know the technical details, you can see the sidebar "How Pages Are Processed on the Server," but that requires some background in object-oriented programming that we won't go into here in detail.

How Pages Are Processed on the Server

When a user enters the URL for a page into a browser, the browser requests that page from the web server. If the page being requested is an *.aspx* page, the server processes the page before returning it.

The *.aspx* page serves as a set of instructions to the server on how to create a standard HTML page to return to the browser. If this is the first time that the *.aspx* page has been requested since the web application started, then the ASP.NET runtime compiles, from the page, a `Page` class that derives from the base `System.Web.UI.Page` class. The compiled class contains all of the control declarations and code that make up the page, including properties, event handlers, and other methods. This compiled class is cached in server memory for faster response on subsequent requests.

In order for an *.aspx* file to be processed by the ASP.NET runtime, it must have a *page directive* as the first line in the file. Directives provide information to the compiler, such as the language in use, the name of the code-behind file, if any, and the name of the `Page` class. Visual Studio automatically generates the page directive for you when you create a new web page. The page directive for your *HelloWorld* web page looks like the following:

```
<%@ Page Language="VB" AutoEventWireup="false"
   CodeFile="Default.aspx.vb" Inherits="_Default" %>
```

With this information, the server is able to run all of the server-side code, translate all of the server-side controls into standard HTML and JavaScript, and assemble an HTML page that will be returned to the calling browser. If the page that is returned includes client-side script, that script will be run on the client-side machine, by that browser, when the page is rendered.

The *.aspx* files can also contain *script blocks*, usually written in JavaScript, to be executed on the client. Server-side code is executed on the server; client-side script is executed on the client's machine by the browser.

The normal structure for an ASP.NET with AJAX application is this: the markup (content) file contains controls. Some of the controls will be server controls. You already saw a server control in the HelloWorld example—`Button1`. The event handler code you wrote for `Button1` was stored in a second file called the *code-behind file*,

and executed on the server. The server controls also contain other code that enables their basic functions, but you usually won't be able to access that. Other controls are AJAX controls, and their code is sometimes written in script blocks in the markup file, or more commonly, it is buried in *.dll* files provided to you by Microsoft (and thus not visible to you as script code), but which is interpreted by the browser on the client machine, so you don't need to worry about how it's done.

There are also HTML controls that are passed "as is" to the client machine's browser, such as tables, and `<br/>` (the line break tag), and so on.

ASP.NET also allows you to create so-called HTML-server controls, which are HTML controls with the tag `runat="server"`, but these are not commonly used, and won't appear in this book.

Once again, this book assumes you have a passing familiarity with HTML, but even if you do not, you should find the examples self-explanatory.

If a markup file is named *Welcome.aspx*, its associated code-behind file will be named *Welcome.aspx.vb*, indicating that the code-behind file is written in Visual Basic (*.vb*) (or *Welcome.aspx.cs* if you are writing in C#).

Controls

As you saw in the Hello World example, *controls* are the building blocks of the web page's graphical user interface (GUI). Some controls that you are probably familiar with include buttons, checkboxes, and listboxes.

Controls allow a user to indicate a preference, enter data, or make selections. They can also provide support for validation, data manipulation, and security, or help to ensure a uniform look and feel to the application.

There are several types of web controls:

HTML controls
: The original controls available to any HTML page, such as `input` (for entering data), `a` (anchor), `div` (for separating and applying format to a section), and more. These all work in ASP.NET exactly as they work in other web pages. HTML controls will be used where appropriate in this book, but will not be discussed in detail. For a good resource on HTML controls, see *HTML & XHTML: The Definitive Guide*, by Chuck Musciano and Bill Kennedy (O'Reilly).

ASP.NET server controls
: Microsoft created the ASP.NET server controls to accomplish two complementary aims. The first was to "normalize" the HTML controls so that the programmer would have a more consistent interaction with the control; the second was to add an extensive and rich set of powerful controls such as data grids, calendars, ad rotators, and more.

ASP.NET AJAX server controls
: ASP.NET AJAX server controls enjoy all the benefits of ASP.NET server controls, such as drag-and-drop operation and a declarative programming model. However, they also include added client-side functionality, helping you to create a smooth and snappy user interface.

User controls and custom controls
: Controls created by the developer or third parties (that is, not Microsoft). This topic is beyond what we'll cover in this book, but for a full discussion of creating these user-defined controls, please see our more advanced text, *Programming ASP.NET* (O'Reilly).

The heart of ASP.NET programming is the ASP.NET server control. With the exception of tables, the traditional HTML controls are replaced by their equivalent ASP.NET controls, both for convenience and flexibility. For example, instead of using a traditional HTML `input` control, you will use instead an ASP.NET `TextBox` control. Not only will this allow the `TextBox` to run server-side code, but it is easier to use, and is more intuitive.

In addition to standard form elements, such as text boxes, labels, buttons, and checkboxes, ASP.NET controls include several broad categories that provide rich functionality with very little code. These include:

Validation controls
: Often, a given field requires a specific format or range of data to be valid. Many of these validation routines are similar and used in many places, such as making sure there is an entry, that two entries match (such as when setting a password), or that an entry falls within a predetermined range of values (which can help protect against certain types of outside attacks). Microsoft provides a full range of built-in validation controls. Chapter 5 discusses these controls in detail.

Data source controls
: Data binding to a variety of data sources, including Microsoft SQL Server and Access, and other relational databases, XML files, and classes implemented in code. Data source controls are covered in Chapter 4.

Data view controls
: Various types of lists and tables that can bind to a data source for display and editing. Data view controls are also covered in Chapter 4.

Login & security controls
: Handle the common chores of logging in to a site and maintaining user passwords. Login and Security controls are covered in detail in Chapter 9.

Personalization controls
: Allow users to personalize their view of a site, including rearrangement of the page itself. User information can be saved automatically and transparently, and retained from one session to the next. Personalization is also covered in detail in Chapter 10.

VB CHEAT SHEET

Classes

Although we've said that you don't need to know much VB to make ASP.NET pages, you need to know a bit of the vocabulary to understand the background discussion in this section. You may have heard that VB.NET is an *object-oriented* language, which means that everything you make with the language is an *object*—every control, every label, even the page itself is an object. Each object is a specific instance of what's called a *class*, or to put it another way, a class is a general case that defines each object. Using the classic example, if `Dog` is a class, then your own dog Sparky is an object—he's an *instance* of the `Dog` class. You can't see or touch `Dog`, but you can see and touch Sparky. In ASP.NET, you can't put the `Label` class on your page, but you can create a `Label` object that's an instance of the `Label` class and put it on your page.

So what's the point of the class, then? The class defines the qualities that the object *has* (called *properties*), and the things that the object *does* (called *methods*).

`Dog` might have properties called `color` and `size`, for example. Each object might have different values for each property, but by definition, they all must have the property. So, Sparky might have a `color` of `brown` and a `size` of `large`, whereas Frisky has a `color` of `white` and a `size` of `small`, but they're both still members of `Dog`. In ASP.NET, a `Label` control has properties for `Text` and `Font`.

Methods, on the other hand, tend to be actions that instances of the class can perform. If `Dog` has methods for `Bark()` and `Eat()`, then both Sparky and Frisky can bark and eat. The `Label` control, for example, has methods of `ApplyStyle()` and `Focus()`, so you can *call* those methods on any `Label` control to apply a style to the label, or set the focus to that control.

You *invoke* both properties and methods with what's called *dot notation*. It's pretty simple; you give the object's name (not the name of the class), followed by a period, followed by the method or property name you want to use, like this:

```
sparky.color = white
lblMyLabel.Text = "The text for the label"
lblMyLabel.ApplyStyle(MyStyle)
```

Although you usually don't see this directly, an ASP.NET page is an instance of the `Page` class. When a user requests an *.aspx* page, ASP.NET creates an instance of a page and then uses the properties and calls methods of that page to generate the HTML that appears in the browser.

Master pages
: Create web sites with a consistent layout and user interface. Master Pages are covered in Chapter 6.

Rich controls
: A subset of ASP.NET controls that implement complex features such as menus, tree views, and wizards.

AJAX controls
: A set of controls that provide special script-based features, including the `ScriptManager`, `UpdatePanel`, `UpdateProgress`, and `Timer`. AJAX is covered in detail in Chapter 3.

AJAX toolkit controls
: An expanding set of controls based on AJAX that provide enhanced client-side functionality without the need to write `JavaScript`, such as watermarks, collapsing panels, and pop ups.

Code-Behind Files

Although you can put your content and your code in a single file, it is strongly discouraged, and we will not do so in this book. The preferred method is to put your content (HTML, server controls, and AJAX controls) into a markup file with the extension *.aspx*, and to put your server-side code into a code-behind file with the extension *.aspx.vb*. You saw this separation of content from code-behind in the Hello World example in Chapter 1.

Events and Postbacks

In the Hello World program you created in Chapter 1, the page was sent back to the server when you clicked the button. When the page returned to the browser, it was displayed with new text, specifically with the words "Hello World."

As we described in that example, clicking on the button raised the `Click` event. It turns out that many controls have a `Click` event, and each control may also have other events specific to itself. For example, list controls typically have an event for when the selected item changes, while text boxes have events for when the text they contain is changed.

The code that responded to the button's `Click` event in Hello World (the control's *event handler*), was a method of the `Page` class, specifically the page that contained the button. As is often the case, this is more confusing to explain than to see in action. To the user, it simply appeared that clicking the button changed the contents of the page.

What is important to keep in mind, however, is that when you click the button, the page is "posted back" to the server. During a *postback*, the browser sends the page to the server, where event handlers are evaluated. The *same* page is then sent back to the browser after the code in the event handlers is run.

When a page is returned to the server and a new page is sent to the browser, that is not a postback. When the page is returned to the server, processing is done, and the same page is returned to the client, *that* is a postback.

Not all controls automatically post back every time you click on them. Buttons do, but just changing the selection in a listbox, for example, normally does not. (You can, if you want, set a listbox to post back every time its selection is changed, as you will see later in this chapter.)

Synchronous and Asynchronous Postbacks

In ASP.NET with AJAX, there are actually two types of postbacks:

Normal

In a normal postback, the entire page is sent back to the server for processing. As just noted, some events do not cause an automatic postback. These events are stored up until a postback occurs, and then they are all handled together. When all of the event handlers have been run, a new HTML page is generated and sent back to the browser.

A normal postback is *synchronous*—nothing else will happen in your application until the server processing is complete and the response is sent back to the browser. The typical time for such an update is less than one second, but this can be dramatically affected by database interactions, network speed, and other factors, some of which are beyond your control. In any event, the user will see the page flicker when the browser redraws it.

Within the normal postback, you often want some behavior to occur the first time the page is loaded, but not on subsequent postbacks. For example, you might want to gather data from a database when the page is first loaded, but not when the user clicks a button to post the page back to the server. There is a property of the `Page`, called `IsPostBack`, which can be tested in your code to determine if this an initial load or a postback. It has the value of `False` the first time the page is loaded, and `True` on subsequent postbacks.

Partial page, asynchronous

AJAX allows an asynchronous postback in which you designate an area of the page to be updated, while the rest of the page remains unaffected. The user usually perceives no page flicker and may be unaware that processing is happening on the server at all. This can make for a dramatically more responsive application.

Be careful with event handling in postbacks. A common bug is caused by assuming that event handler A will run before event handler B. The best way to discover such bugs is by using the debugger to examine the code as it runs, which we will explore in Chapter 8.

The next example will demonstrate a normal postback and an AJAX asynchronous (partial) postback with three labels: one will update the first time the page is loaded, one will update only on a subsequent normal postback, and one will update asynchronously.

To start, create a new web site named Postbacks. If necessary, refer back to Figure 1-2, and create the new web site just as you did in the Hello World example. Be sure to name the web site folder *Postbacks*, so that the site will be named *Postbacks* as well. When the project is open, switch to Design view by clicking on the Design tab at the bottom of the editing surface.

To make this an AJAX-enabled web page, it must have a `ScriptManager` control. Expand the AJAX Extensions section of the Toolbox and drag a `ScriptManager` control anywhere on the page. We'll discuss the `ScriptManager` in detail in Chapter 3, but be assured that it will not be visible when your application is running; its job is to work behind the scenes to coordinate the AJAX controls on the page.

Press the Enter key once to move the cursor below the `ScriptManager` control, then type in the text:

```
Page First Loaded at:
```

Drag a `Label` control from the Toolbox onto the design surface next to the text you just typed. Click on the `Label` control to select it, so that the Properties window shows the properties for the label. The Properties window is on the lower right of the interface. If you don't see it, press F4 to get it to appear. You'll know if you've selected the right control because its name will be listed at the top of the Properties window—in this case, `Label1`. Before proceeding, you should change the ID of the `Label` to something more meaningful—in the Properties window, scroll up or down until you find the (ID) property in the left column. Click in the right column, delete `Label1`, and then type in `lblFirstPageLoad`. Now find the `Width` property in the left column, and change its value to `200px` in the right column.

Drag a button onto the page to the right of the label. Select the button, and in the Properties window, change the button's ID to `btnPostback` and the `Text` property to `Postback`, in the same way that you changed the label's properties.

With the cursor just to the right of the `Button` on the design surface, press the Enter key to move the cursor to the next line. Type in the following text:

```
Page posted back at:
```

Drag another `Label` control onto the page, to the right of the text you just typed, and set its (ID) property to `lblPostBackPageLoad`.

Your page should now look pretty much like that shown in Figure 2-1.

In Solution Explorer, click on the plus sign next to `Default.aspx` to expand the list of files. You will see the code-behind file created for you, named `Default.aspx.vb`. Double-click on the code-behind file to display that file in the editing window, as shown in Figure 2-2.

VB CHEAT SHEET

Namespaces

When you create your pages, you'll use lots of names. Every control on the page gets a name, as do all the properties of those controls. When you write event handlers in the code-behind file, you may create names for objects or variables in your code as well, even though the user of your page will never see them. With all those names, it's possible that you might inadvertently pick a name that has already been used by Microsoft or by the creator of a different page in your site. To avoid that problem, ASP.NET uses the concept of *namespaces*. Simply put, a namespace is a way of saying. "These names belong to this group." One of the namespaces you'll see all the time is the `System` namespace. This namespace appears in all your event handlers as part of `System.EventArgs`. This indicates that the `EventArgs` object belongs to the `System` group of names, which is a special namespace that .NET reserves for its built-in objects. You won't be creating your own namespaces in this book, but you should know what they are when you see them.

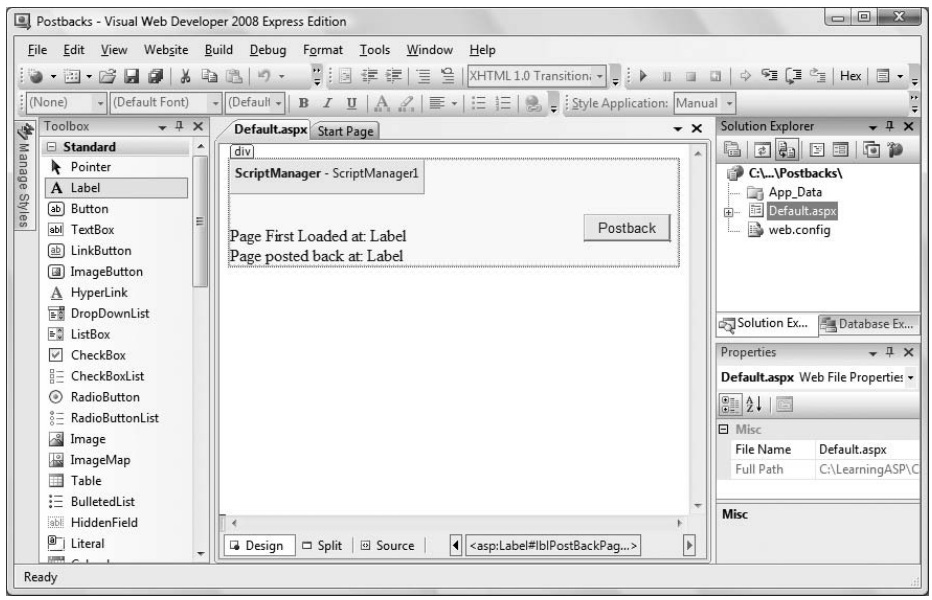


Figure 2-1. *Drag two Labels and a Button onto the Postbacks page and set a few properties.*

You can also open the code-behind window by right-clicking on the markup window and choosing "View Code."

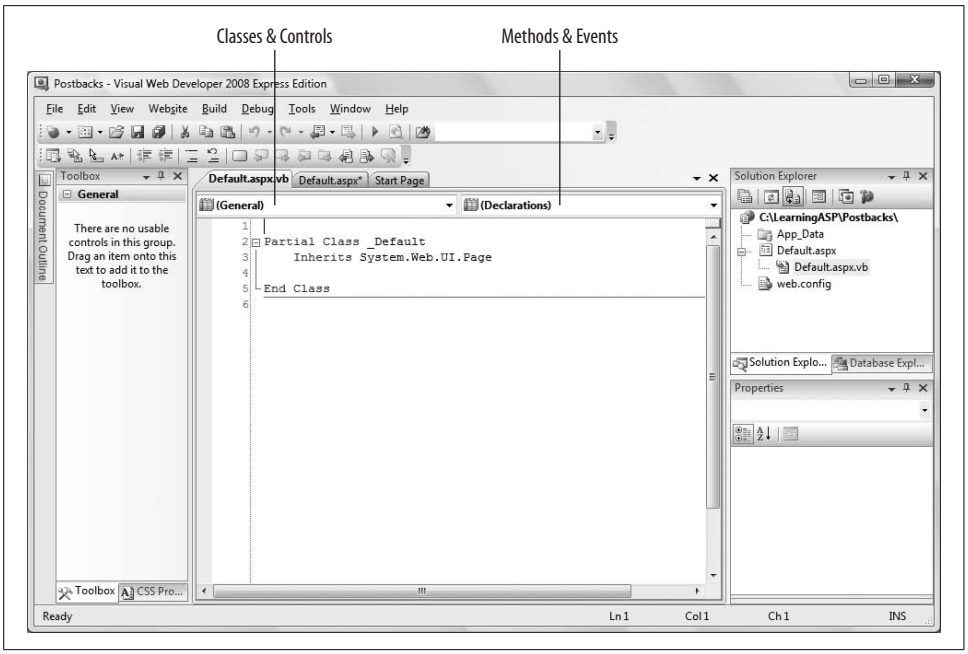


Figure 2-2. *The code-behind editing window for the Postbacks page, showing the Classes & Controls and Methods & Events drop-downs. Set the Classes & Controls method to (Page Events), and the Methods & Events drop-down to Load.*

Naming Conventions

Microsoft's .NET naming guidelines prohibit the use of Hungarian notation for all "public" identifiers. Hungarian notation is the practice of prepending variable names with letters that indicate the type of the identifier (e.g., prepending a variable of type integer with "i"). (You can read about the history of Hungarian notation at *http://en.wikipedia.org/wiki/Hungarian_Notation*.)

Because the guidelines do allow the use of Hungarian notation in private member variables, two schools of thought have arisen about using this notation, especially when referring to controls on a page. Many developers will refer to a text box, for example, as `txtLastName`, while others will name the same text box `LastName`. The authors of this book represent vociferous advocates of both camps. (In fact, one of the authors is a strong advocate on both sides of this issue depending on his mood.) You will, therefore, stumble across both notations in this book. This is not a bug, it is a feature, intentionally included to help you become used to both approaches. Honest.

The Page Load event and synchronous postback

Every time an ASP.NET page is displayed, the Page's `Load` event is fired. You'll use that event to display the time that the page was initially loaded and posted back. To do so, you need to create an event handler for the Page `Load` event.

At the top of the code window are two context-sensitive drop-down controls, as indicated in Figure 2-2.

The drop-down on the left displays the classes and controls in your application (as well as the useful all-purpose setting [General]); the one on the right displays all the methods and events for the class or control selected on the left.

The IDE is slightly different here if you're working in C# rather than VB.NET. The drop-downs at the top of the editing window are slightly different, and in any event, the empty `Page_Load` event handler will already be created on the page for you.

Select (Page Events) in the left drop-down, and select Load in the right drop-down. This will bring up a code skeleton for the `Page_Load` event handler, shown in Figure 2-3.

Type the highlighted code from Example 2-1 into the `Page_Load` event handler. Notice how IntelliSense helps you with the name of the control and its available properties.

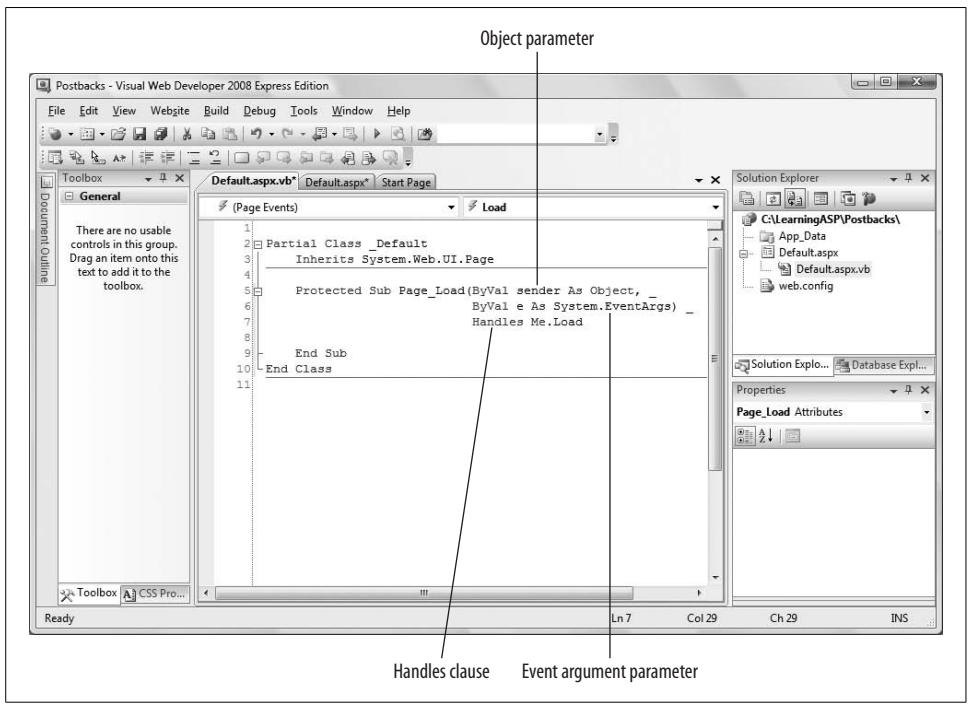


Figure 2-3. *After you select Page Events and Load from the code-behind file drop-down, a code skeleton for the Page_Load event handler is inserted automatically.*

Example 2-1. *Page_Load for Postbacks example*

```
Protected Sub Page_Load( _
              ByVal sender As Object, _
              ByVal e As System.EventArgs) _
              Handles Me.Load
    lblPostbackPageLoad.Text = DateTime.Now

    If (Not IsPostBack) Then
        lblFirstPageLoad.Text = DateTime.Now
    End If
End Sub
```

Note to C# Users: In VB.NET. the line of code in Example 2-1 that assigns the date and time to the Label `Text` property implicitly converts `DateTime.Now` from a `DateTime` type to a `String` type. In C#, this implicit conversion does not occur, so you must use the following line of code instead:

```
lblPageLoad.Text = DateTime.Now.ToString( );
```

VB CHEAT SHEET

Booleans

The *Boolean* variable is a special type of variable that can have only one of two values: `true` or `false`. They're very useful for evaluating simple conditions, and taking an action based on whether the condition is `true`. `IsPostBack` is a Boolean variable, and has the value of `True` if the page loads as the result of a postback. The `Not` keyword is used to indicate the *opposite* of whatever Boolean value follows, so in this case, `Not IsPostBack` is `True` if the page is loading for the first time. In this case, the value of the Boolean is used to control the behavior in the `If` statement. You'll see this technique used a lot.

Run the application. If it is the first time the web site has been run, the IDE will offer to modify the *web.config* file to enable debugging. Click OK to that. A browser will open with text similar to that shown in Figure 2-4.

Buttons post back to the server even if you do not implement an event handler for their `Click` event. Click the button a few times. As you can see, each time you click the button, the page is posted back. You will see the second `Label` updated each time, but the first `Label` still shows only the time the page was first loaded.

Take a look at the status bar to see the change to the page being sent back to the "server" (in quotes because in this case the server is your local machine). For each postback, the page is reloaded, triggering a `Page_Load` event. This in turn causes the `Load` event handler to run and the `Label`'s text to be updated with the current time.

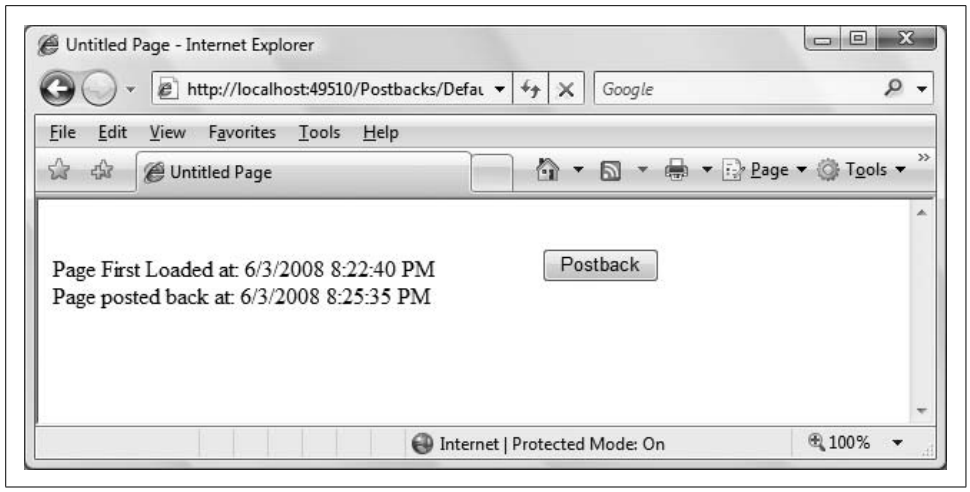


Figure 2-4. *When you run the Postback page for the first time, the times will be identical. After you click the button, the lblPostBackPageLoad Label will update, but the lblFirstPageLoad Label will not.*

The `if-then` block in the `Load` event handler tests the value of `IsPostBack`. If it is `False`, that is, if it is the first time the page is loaded, then the first `Label` is updated. On subsequent postbacks, that line of code will not be reached, and so that `Label` is not updated. However, the second `Label` is updated every time.

If you are adventurous, you can put a break point in the `PageLoad` event handler. Open the code behind file, navigate to the `Page_Load` method, and then click in the left margin next to the line with the `lblPageLoad.Text` instruction. If a red dot appears in the margin, you'll know you've done it right. Press F5 to run the app in debugging mode. You will see this break point stop the application each time the page is about to run. We will cover debugging in detail in Chapter 8, but we couldn't resist showing you that this really works. After you're done, click on the red dot again to remove the break point.

You can achieve the same result by clicking the refresh button on your browser, which forces a refresh of the current page, and thus a post back to the server. When you're done, close your browser to stop the application.

Adding asynchronous postbacks

With traditional postback code in place, you'll modify this application to add the ability to make an *asynchronous* postback using AJAX.

Return to *default.aspx* by double-clicking on it in Solution Explorer, then click the Design tab to switch to Design view. Bring up the Toolbox and pin it in place. Expand the AJAX extensions section in the Toolbox. Place the cursor after the

VB CHEAT SHEET

Methods, Event Handlers, Parameters, Arguments

In Visual Basic.NET, a method is implemented either as a sub (which returns no value) or a function (which returns a value).

Methods may declare values that are *passed into* the method and used as if they were declared as local variables. In VB.NET, you must declare the type of the value to be passed in. When the method is called, you must pass in a value. The declaration and the value passed in are called either *parameter* or *argument*. Some old-school computer scientists distinguish between these terms, but we will use them interchangeably.

Event handlers are special methods that run when an event is raised by a control, or by the operating system, or by something else happening in your program that requires a response. By convention, event handlers in ASP.NET always take two arguments. The first, marked as type `object`, is the object that caused the event, and is named, by convention, `sender`. The second is marked as type `EventArgs` or a type that derives from `EventArgs`. (Derivation is a concept from object-oriented programming. It can be summarized as "specialization." When you derive a `Cat` from `Animal`, you say that a `Cat` is an animal, but a special type of animal, with special characteristics or special behaviors.) This second parameter, the `EventArgs`, is usually called `e`.

Event handlers in VB.NET are linked to the event that they handle with the `Handles` keyword. Thus, you might mark the method `Page_Load` with the keywords `Handles Load` or `Handles Me.Load` where `Me` is a keyword that refers to the object itself (in this case, the page) signaling .NET that this method handles the `Load` event of the page.

`lblPostBackPageLoad` control and press the Enter key to move the cursor down to the next line. Drag an `UpdatePanel` from the Toolbox onto the design surface.

The AJAX `UpdatePanel` control is the key to asynchronous updates. Any controls that you place within the `UpdatePanel` will be updated asynchronously, including both standard ASP.NET and HTML controls. The panel acts as an asynchronous portal back to the server.

To display the time the `UpdatePanel` was updated, add the following text inside the `UpdatePanel`:

```
Partial-Page Update at:
```

Scroll back up within the Toolbox to the Standard controls and drag another `Label` control into the `UpdatePanel`. Be sure this control is inside the `UpdatePanel` or this example won't work. Make sure the new label is selected, and update its properties. Change its ID to `lblPartialUpdate` and set its Width property to `200px`.

VB CHEAT SHEET

If-Then Statements

When you're working with data, you usually don't know what the data will be when you're writing your code. You might want to take different actions depending on the value of a variable. That's what the `If-Then` statement is for:

```
If chkMyCheckBox.Checked = True Then
  txtMyTextBox.Text = "It's true!"
End If
```

The condition you want to evaluate comes after the `If`, but before the `Then`. In this case, you want to determine if the checkbox is checked, so the condition is `chkMyCheckBox.Checked = True`.

If it's true, the statement after the `Then` keyword is executed, setting `txtMyTextBox.Text` to "It's true!" You can execute any number of statements in the `Then` section.

If the condition is false, nothing happens.

You must insert the statement `End If` at the end of the `Then` block so that your code knows where the `Then` block ends and can continue executing as normal from that point.

The `Else` statement comes into play when you want to take one of two actions. With just an `If-Then` statement, if the condition you're evaluating is false, nothing happens. However, you might want to take one action if the condition is true, and another if it's false, like this:

```
If chkMyCheckBox.Checked = True Then
  txtMyTextBox.Text = "It's true!"
Else
  txtMyTextBox.Text = "Not true!"
End If
```

This code sends one message to `txtMyTextBox` if the condition is true, and a different message if it's false.

You have lots of options when you specify conditions, which are based on a set of *operators* that you're probably already familiar with. For example, instead of testing to see if one part of your condition is equal (`=`) to another, you could use one of these other operators:

- `<>` not equal to
- `<` less than
- `>` greater than
- `<=` less than or equal to
- `>=` greater than or equal to

—continued—

In short, you can test for any condition that evaluates to true or false—in other words, a Boolean. In fact, the `Checked` property of a `TextBox` is a Boolean all by itself, so you could have used this for the condition:

```
If chkMyCheckBox.Checked Then
```

You can also add these controls in Source view, either typing the code by hand, or by dragging a control from the Toolbox. You can then set properties in the Properties window or type in attributes directly in the code window.

Drag a `Button` control into the `UpdatePanel`. In the Properties Window, change the ID of that `Button` to `btnPartialUpdate`, and set the `Text` property to "Partial Update." The Design view should look something like Figure 2-5.

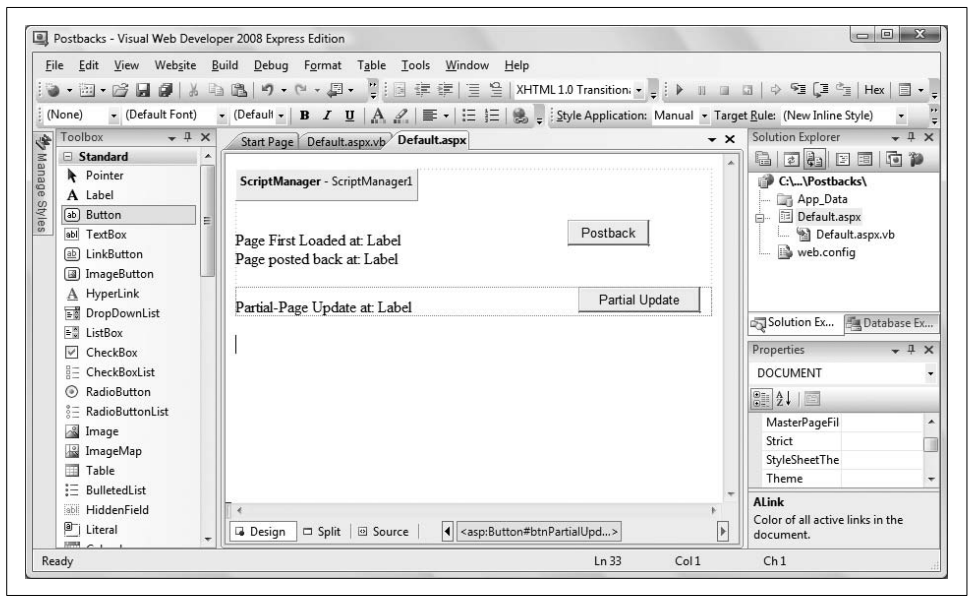


Figure 2-5. *After you've added the UpdatePanel and the Partial Postback controls, your Design view should look like this.*

Next, you need to add an event handler for `btnPartialUpdate`'s `Click` event. Double-click on `btnPartialUpdate`. The default event for a button is its `Click` event, and when you double-click on a control, the default event handler is created for you. The code behind file will open within the default event handler. As you have already seen, the event handler gets its name from the control and the event, separated by an underscore, in this case `btnPartialUpdate_Click`.

Enter the highlighted line of code from Example 2-2 in the click event handler for `btnPartialUpdate`.

Example 2-2. *Click Event Handler for btnPartialUpdate*

```
Protected Sub btnPartialUpdate_Click( _
          ByVal sender As Object, _
          ByVal e As System.EventArgs)
    lblPartialUpdate.Text = DateTime.Now
End Sub
```

Once you've made the change, run the updated application. After the page loads, click each of the buttons a few times. You will see something similar to Figure 2-6.

Depending on which region your computer is in and how your region options are set, you may see the date and time displayed using a different format.

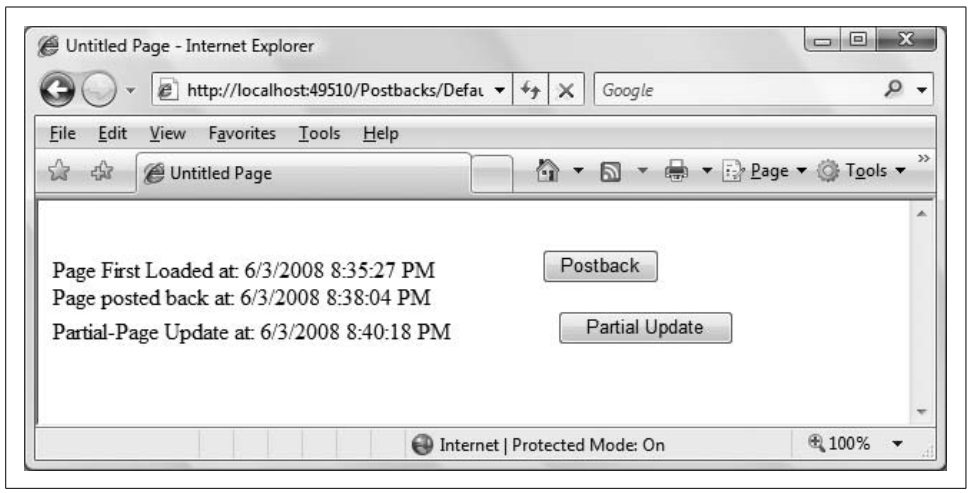


Figure 2-6. *After you click the Partial Update button, the label in your UpdatePanel refreshes, but the labels outside the UpdatePanel do not. Likewise, the Postback button only refreshes the postback label, not the label inside the UpdatePanel.*

Note the following results:

- Clicking on the Partial Update button updates the label in the `UpdatePanel` control, but not the labels outside the `UpdatePanel`.
- Clicking on the Postback button updates only the postback label outside the `UpdatePanel`, but not the first load label, nor the label inside the `UpdatePanel`.
- The `UpdatePanel` is invisible to the user (though its effects are not).

When you're done, close the browser.

What's great about updating just a portion of a page this way is that it not only eliminates "flicker," but your entire application will *seem* faster and more responsive to the end user.

Silverlight

In March of 2007, Microsoft unveiled Silverlight 1.0 in Beta and started a new chapter for .NET programmers. That chapter is still being written as this book goes to print, though with the release of Silverlight 2.0 Beta 1 in March of 2008 the shape of Silverlight and its potential impact is much easier to see.

There are now five distinct technologies within .NET for writing applications, and they form a spectrum from entirely server-side to entirely client side:

- ASP.NET
- ASP.NET with AJAX
- Silverlight (potentially integrated with ASP.NET or ASP.NET with AJAX)
- Windows Forms
- Windows Presentation Foundation (WPF)

The sweet spot for most developers is in the center: applications delivered through the browser, but ones that also provide a very rich, interactive, responsive experience for the user. Microsoft now offers two technologies to accomplish this: AJAX and Silverlight.

AJAX is, in truth, not a new technology at all; it is the aggregation of mature technologies in new ways (JavaScript and XML delivered asynchronously) supplemented by some very cool well-tested code written by Microsoft (the AJAX libraries) and others.

AJAX has the advantage in that it runs on any browser that supports industry standard protocols.

Silverlight, on the other hand, requires a browser plug-in that the user has to accept. That plug-in contains a carefully chosen subset of the CLR, allowing Silverlight applications to use managed code languages like VB.NET or C# and to implement applications built on a subset of XAML, the mark up language used by WPF (and Work Flow).

The net result is that Silverlight applications are *much* faster, richer and have capabilities that are simply not possible with AJAX, but they are limited to running (at the time of this writing) on IE, Firefox, Safari on Windows, the Mac, and Linux.

Learning Silverlight is not hard, but there is quite a bit to it. A full discussion would be a book in itself (see *Programming Silverlight* by Jesse Liberty and Tim Heuer [O'Reilly]).

There may come a time when Silverlight is as much a part of every ASP.NET programmer's skill set as AJAX is today, but we're not quite there yet, and to keep this book at a manageable size, we've decided to defer a full discussion of Silverlight for now. To learn more, take a look at *Essential Silverlight* by Christian Wenz (O'Reilly) and the extensive material available at *http://Silverlight.net*.

Using Controls

As you've seen in both examples so far, when you drag a control from the Toolbox onto the design surface, it is generally represented as a visible widget to the user. Some controls, however, are used not for display, but for manipulating other objects (for example, database manipulation controls), and these are displayed in a special area at the bottom of the main window.

In any case, every control is identified by a unique `ID` property. Both Visual Web Developer and Visual Studio will automatically assign an ID to your control as you drag it onto your page. These automatically generated IDs are rarely meaningful, and we suggest that you rename them. For example, while the IDE might name your label "Label2," you will probably find it much more useful to rename that label something like `lblPartialUpdate`.

When you click on a control in Design or Source views, its properties are shown in the Properties window. You can change any property value in the Properties window or directly in Source view, and any changes you make will be reflected in both places immediately.

Organizing the Properties Window

Within the Properties window, you can group properties by category or alphabetically. Figure 2-7 shows the Accessibility, Appearance, and Behavior categories of a button, though there are others. You can click the appropriate buttons in the menu bar to toggle between the Categorized and Alphabetical views. (When organized alphabetically, the ID of the Control is placed at the top of the list, out of order, for convenience.)

Virtually every control has events associated with it. To view a control's events, click the lightning bolt button. To switch back to properties click the Properties button.

Finding properties with IntelliSense

You don't need to use the Properties window to edit control properties if you don't want to—you can type the properties and their values directly into Source view. If you prefer to work in Source view rather than Design view, you can enlist IntelliSense to help you find both the properties and events for any given control. With the cursor inside the markup for a given control, as you press the Space bar, the list of members for the control will be displayed. You can then select a member or value from the list. If you change your mind and don't want to use the IntelliSense list, press Esc to dismiss it. As you type, IntelliSense will help you fill in the appropriate property or event, as shown in Figure 2-8.

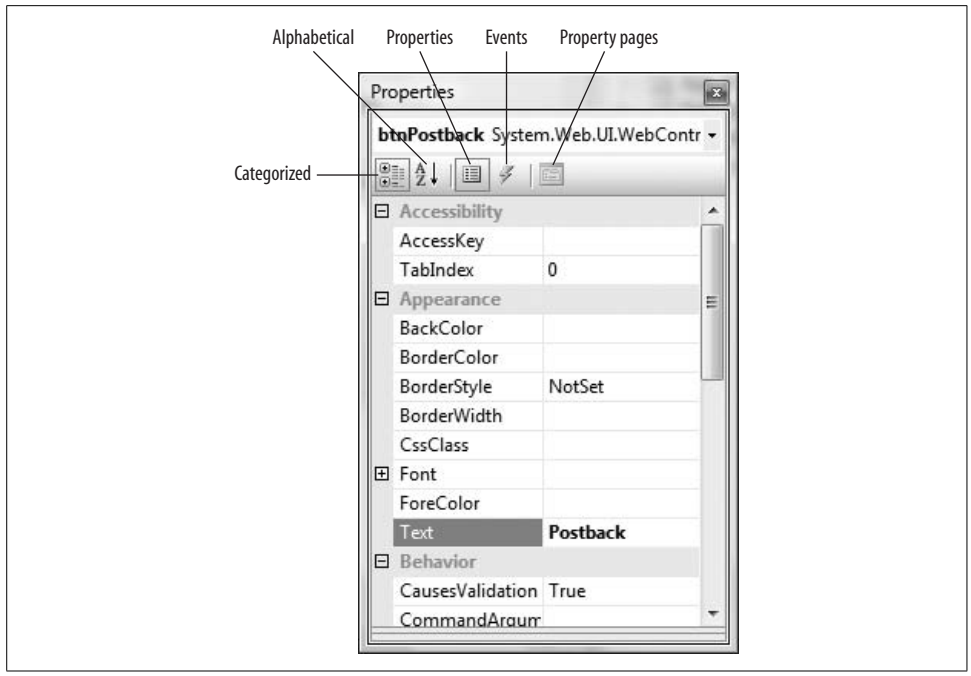


Figure 2-7. *The Properties window, as you would expect, shows you the properties of the control you select. You can organize the properties by category, as shown here, or alphabetically. You can also view the events associated with the control.*

Basic Controls

We could simply review the basic controls in a vacuum, but that's not very interesting. Instead, in this section you'll create a simple business application using the ASP.NET and AJAX controls in context. The application will be for a fictitious company called AdventureWorks, a recreational equipment retailer.

To begin, close the Postback site, if it's still open (File→Close Project), then create a new web site using the ASP.NET Web Site template (similar to what you did in the previous example). Name your new project AdventureWorks.

For this version of the program, you'll use hardcoded data. In later chapters, you will add dynamic content with data retrieved from a database. Also, to keep this simple, you will not initially use any AJAX features; that will be added in the next chapter.

The first page you're going to build is the order form. The finished page will look something like Figure 2-9, where all the types of controls are labeled. This somewhat contrived web page (see the upcoming sidebar "Good Sites Look Good") is intended to demonstrate many of the available ASP.NET controls for various applications.

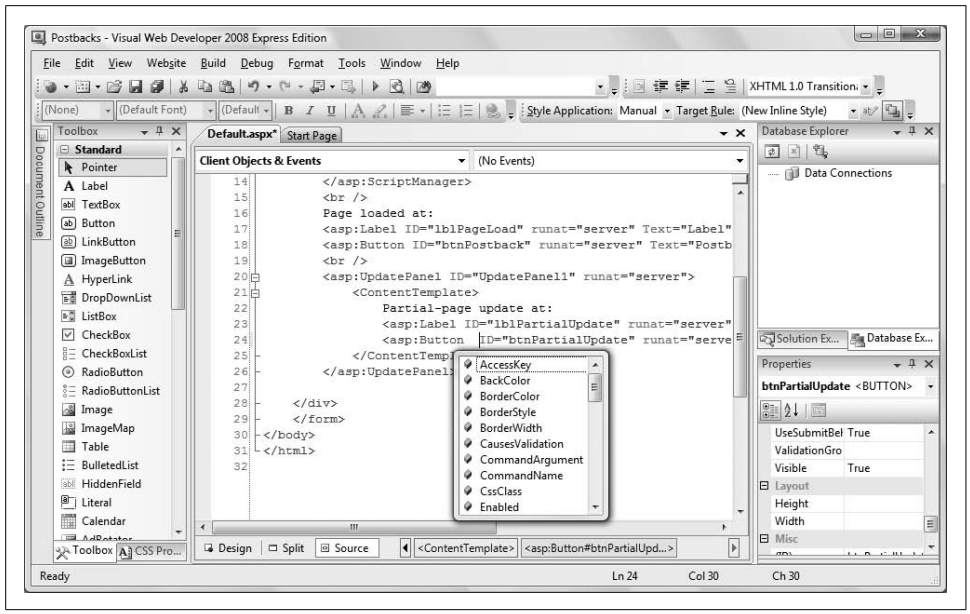


Figure 2-8. *IntelliSense provides a drop-down as you type, so that you can select the property or event you want to use.*

In Solution Explorer, rename *default.aspx* to *OrderForm.aspx* by right-clicking on the file name and selecting Rename. The code-behind file is automatically renamed, as well as *almost* all the internal references.

When you use automatic renaming, be careful about names that are used in text or in page directives (at the top of HTML files)—they will not be renamed for you. Also, the name of the class in the code-behind file will not be updated automatically.

In this example, you'll work in Design and Source views, moving back and forth depending on which is most convenient for the task at hand. Feel free to use Split view to see both views at once.

Open *OrderForm.aspx* and select Source view. Change the text between the `<title>` tags from `Untitled Page` to `AdventureWorks`, and then run the application. At this point, an empty browser will come up with `AdventureWorks` in the title bar.

That was fun. Now, add some substance to the page, beginning with header text. Close the browser. In Source view, type in the following HTML between the `<div>` tags:

```
<h1>AdventureWorks Order Form</h1>
```

Notice how IntelliSense helps by entering the closing tag for you.

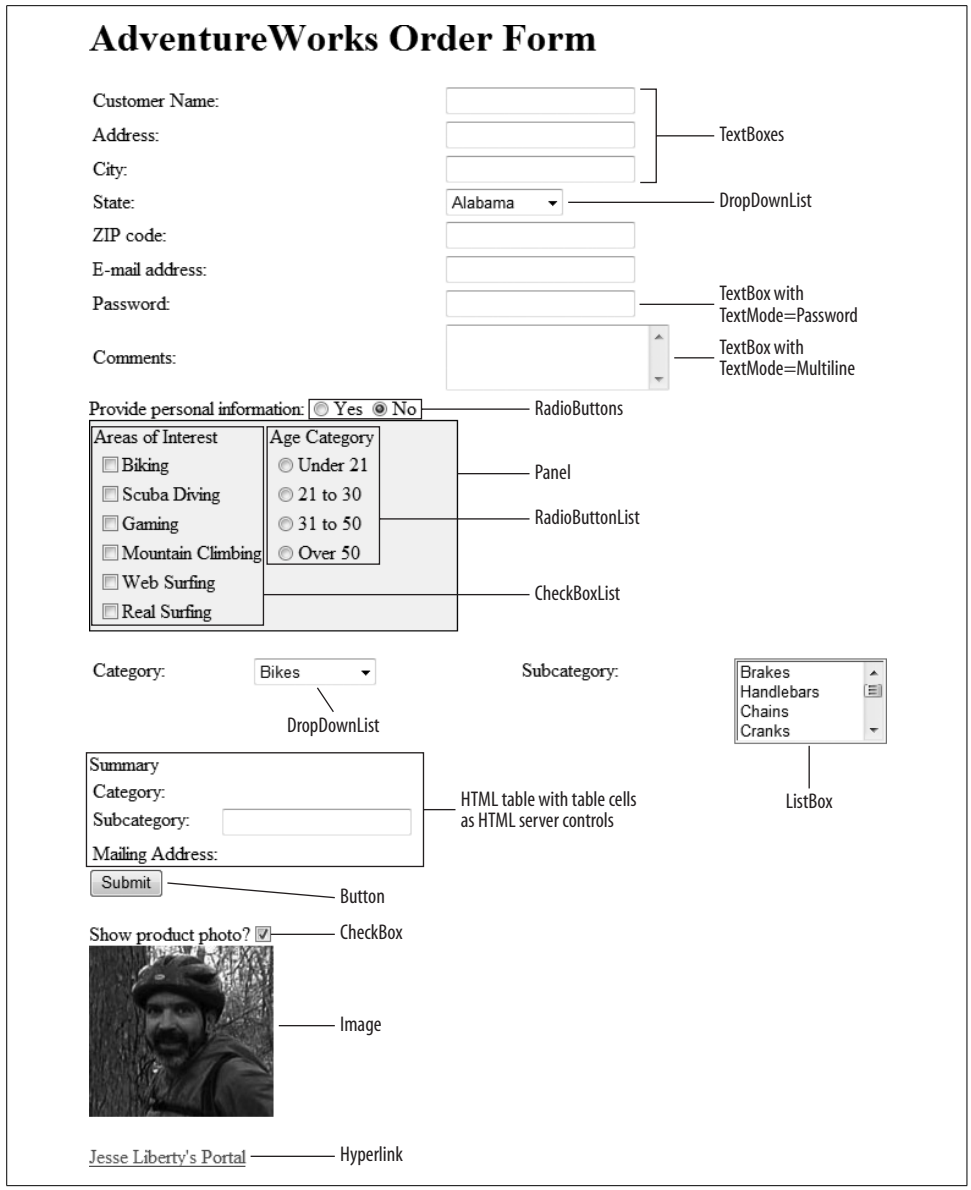


Figure 2-9. *This is what the AdventureWorks Order Form in this example will look like when it's finished. It's not the prettiest page, but it uses lots of the controls you'll be using in this book.*

Alternatively, in Design view, you can just type in the text on the design surface and then highlight the text and click on the Block Format drop-down menu in the Formatting toolbar, as shown in Figure 2-10.

Good Sites Look Good

Our only excuse for how ugly the forms in this book are is that we are consciously avoiding all styling to keep the examples as simple as possible.

We *do* believe, however, that the creation of professional quality web sites requires going beyond just the programming, and includes creating professional looking websites as well.

Unfortunately, using style sheets, let alone image buttons, images, gradients, and all of the other ingredients necessary for a truly professional looking web site would clutter up the examples in the book, making it longer than necessary, and would only get in the way of what you really want to learn about—ASP.NET.

We will return to style sheets in Chapter 7, and we do cover all of these subjects in some detail in *Programming ASP.NET*. However, the art and skill of creating truly professional looking web sites requires many books, as this approaches an art form.

Among the resources we recommend are the highly acclaimed web site *http://www.csszengarden.com/*, and its associated book, *The Zen of CSS Design: Visual Enlightenment for the Web* by Dave Shea and Molly Holzschlag (Peachpit), as well as the following books:

- *The Non-Designer's Design Book* by Robin Williams (Peachpit)
- *Don't Make Me Think*: *A Common Sense Approach to Web Usability* by Steve Krug (New Riders)
- The complete Classroom in a Book series by Adobe
- The Total Training computer-based courses: Adoebe Creative Suite 3 and Adobe Macromedia Studio 8 bundles

For other recommendations, please go to *http://www.LibertyAssociates.com* and click on "Books," then click on "Recommendations," then "Technical and Programming."

Creating Tables

To position the controls on the page, you'll need to create a table. If you're comfortable with HTML, you can certainly insert your table row and column tags manually in Source view and receive assistance from IntelliSense as you go. If you prefer, however, VS and VWD offer an insert table wizard.

To see this at work, switch to Design view (some of the procedures that follow cannot be done in Source view), position the cursor immediately after the heading you just entered, and press Enter once to start a new line. Click on the Table → Insert Table menu item to bring up the insert table dialog box and enter, for this example, eight rows and two columns, as shown in Figure 2-11.

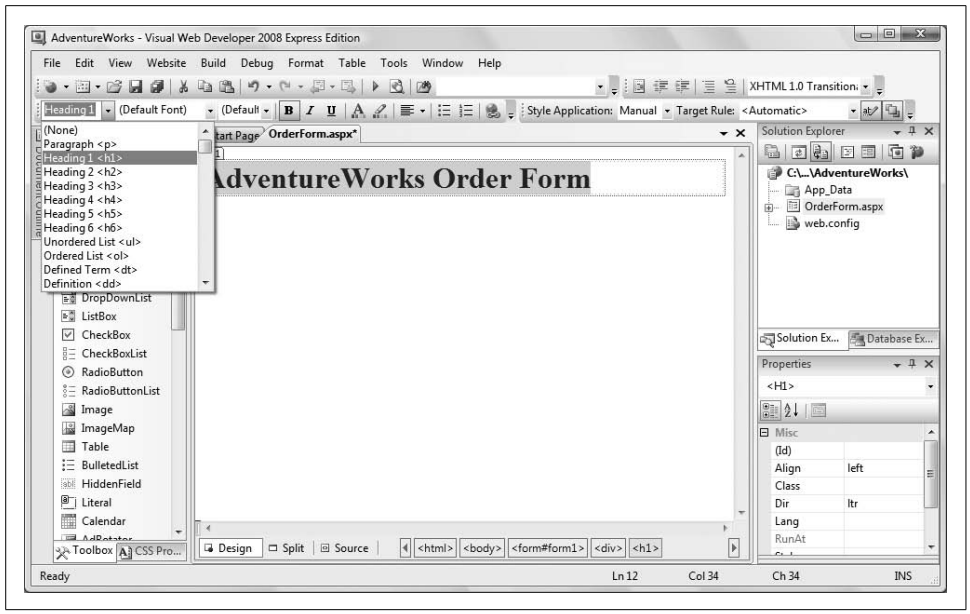


Figure 2-10. *You can enter this heading in Source view, or you can enter it in Design view, and apply the formatting with the Block Format drop-down menu.*

Figure 2-11 demonstrates how you can use this dialog to set various attributes for the table, although you can also adjust these attributes later in the Source view. Click OK to create the table.

You'll use this table to align all of the prompts in the left column, and the user input in the right column. As is typical with most ASP.NET pages, you'll use HTML to generate the display text for your prompts, and ASP.NET `TextBox` controls for most of the user input. Figure 2-12 shows the end result for this portion of the page.

Type in the text shown in the left column in Figure 2-12, and then add the controls to the right column. For the Customer Name, Address, City, ZIP code, and E-mail fields, the controls are simple `TextBoxes`, so you can just drag them from the Toolbox into the appropriate table cells. The Password and Comment fields are special `TextBoxes` that we'll cover in the next section. The State field is a drop-down list that we'll get to a bit later in this chapter. You can leave those cells empty for the moment.

Every ASP.NET control must have a unique ID, and when you drag your text boxes onto the page, the IDE will assign a simple default ID for you. We strongly recommend, however, that you rename each text box with a meaningful name to make your code easier to read and maintain. It is far easier to understand code that refers to `txtName` than code that refers to the same field as `TextBox1`. Name the `TextBoxes` (in order): `txtName`, `txtAddress`, `txtCity`, `txtZip`, `txtEmail`, `txtPassword`, and `txtComments` (skipping the `DropDownList`).

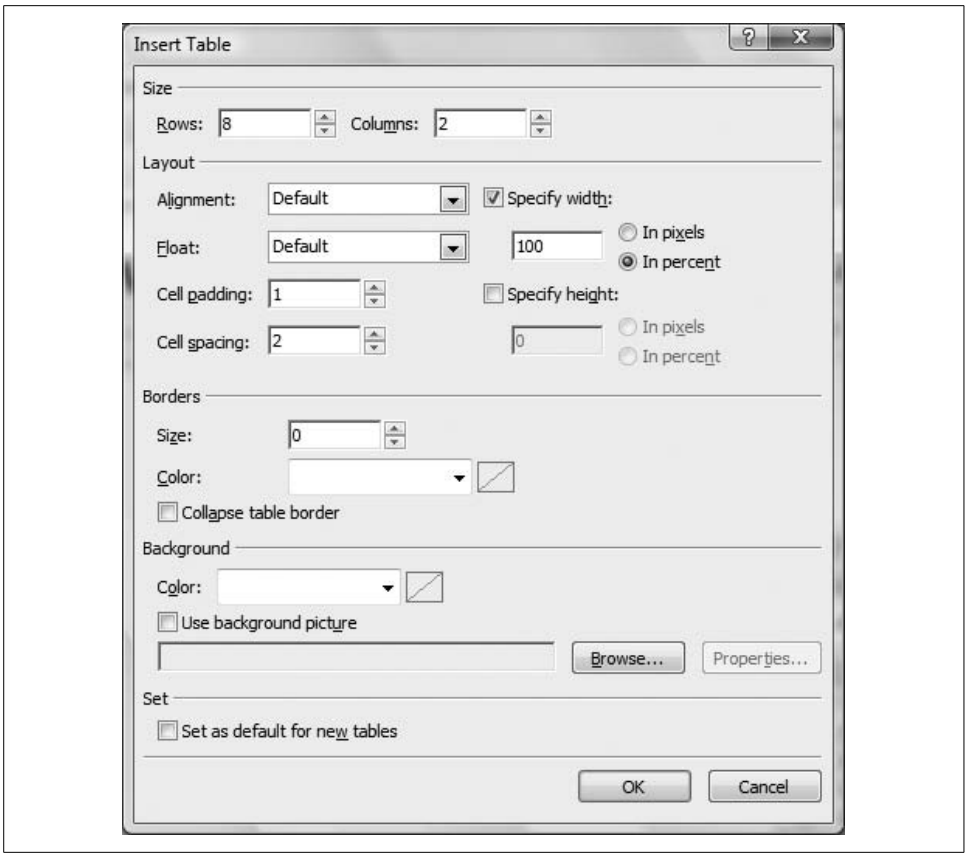


Figure 2-11. *For this example, enter 8 rows and 2 columns in the Insert Table dialog box.*

Setting Properties

There are four ways to set the properties of your controls: programmatically, in the markup, in the Properties window, or through a wizard.

For example, the `TextBox` control has a `Text` property. You can set this property either declaratively in your markup or programmatically in your code-behind file. You can also read from that property programmatically. You might, for example, write:

```
Dim City As String
City = txtCity.Text
```

You can set the font characteristics for text in a text box in the markup, programmatically, or in the Properties window, as shown in Figure 2-13.

The `TextMode` property for text boxes allows you one of three settings: `SingleLine`, `MultiLine`, or `Password`. If you choose `Password`, the text that is entered will appear as dots. Select the `TextBox` for the password, and change the `TextMode` property to `Password`.

Tables, Page Layouts, and the HTML/CSS Debate

To lay out an *.aspx* page with the necessary precision, you have a number of options. The two most common and successful of methods are to use either HTML tables or Cascading Style Sheets (CSS).

Many CSS aficionados believe that HTML should only be used to describe "content," and cascading style sheets should be used to describe layout. It would be their position that HTML tables should be used only to create tabular data, and never as a tool for manipulating the layout of the page. Certainly it is true that when using HTML tables for layout, you will find yourself forced to use "nested tables," that is, tables within tables (within tables, ad infinitum) to get the level of precise control your page might need. It is argued that this is not only inefficient but difficult to maintain.

Whatever the theoretical or practical merits of this argument, few would disagree that the use of HTML tables for layout is a well-established tradition, and is certainly easier to demonstrate than using CSS. In any case the layouts we will be using for our sample applications will be simple enough that we will be satisfied with HTML tables for layout.

ASP.NET provides an ASP.NET Table control, which you can drag onto your form from the Toolbox. We believe it is more inconvenient than it is worth though because it does not size properly in design view and using HTML tables is generally easier, faster, and less resource-intensive on the web server.

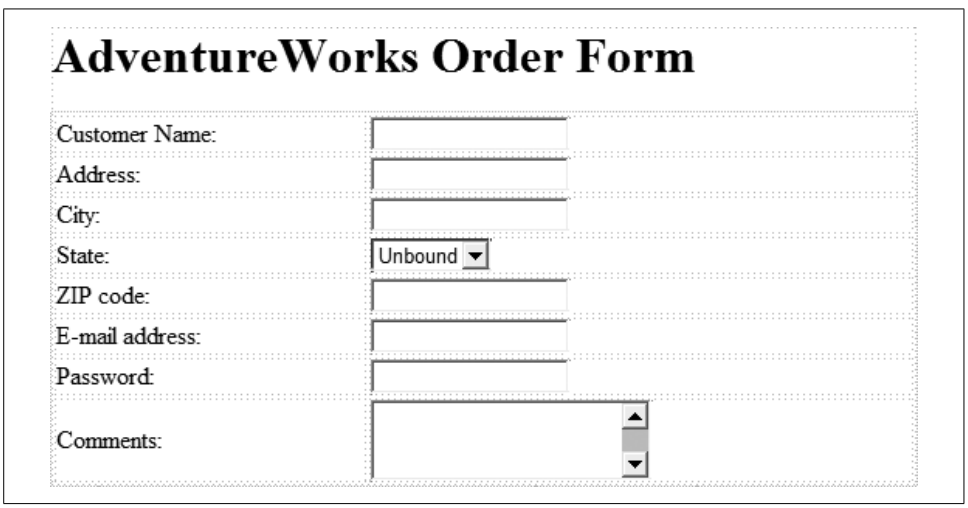


Figure 2-12. *You'll use a two-column table to hold the user prompts and input fields in this example.*

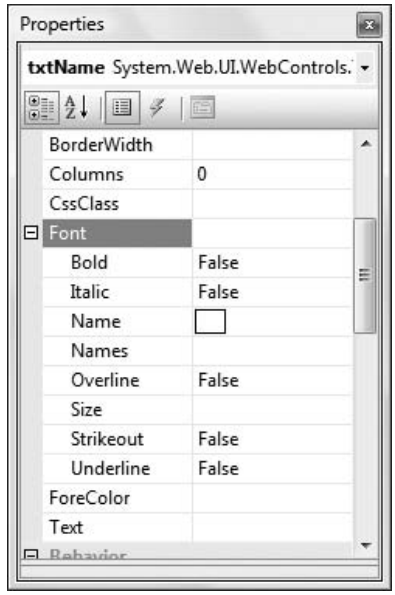


Figure 2-13. *Use the Font section of the Properties window to set the font characteristics of the TextBoxes in your page.*

ASP.NET controls treat the font family, or individual character attributes such as bold, as a property of the `TextBox` class, while for HTML controls, it would be more typical to use styles, set most typically from a style sheet. We cover style sheets in Chapter 6.

Now, set the `TextMode` property on the comment text box to `MultiLine`. Set the `Rows` property to 3 to create a three-line comment field. Run your application again, and try typing in the Password and Comments fields to see how these special text fields work.

Selection Controls

ASP.NET offers a number of different controls to create lists from which the user can make a selection. These include the *ListBox*, the *DropDownList, RadioButtons* and *RadioButtonLists, CheckBoxes*, and *CheckBoxLists*. All of these controls work more or less the way you'd expect them to.

While not used for selection, ASP.NET has one more kind of list used for organization: the *BulletedList*. `BulletedLists` have a `BulletStyle` property, which can be set to numbered, lower- or uppercase alphabetic, lower- or uppercase Roman numeral, disk, circle, square, or a custom image.

Radio button lists and checkbox lists are convenient for creating and grouping more than one radio button or checkbox at a time. Table 2-1 reviews the use of each of these different types of selection controls.

Table 2-1. *Summary of selection controls*

Control type	Selection	Best for?
CheckBox	Multiple	Short lists
CheckBoxList	Multiple	Short lists
RadioButton	Single	Short lists
RadioButtonList	Single	Short lists
DropDown	Single	Long lists
ListBox	Multiple	Long lists

Referring back to Figure 2-9, just below the table that gathers the user's name and address, you want to add a control to prompt the user to decide whether to provide certain personal information. Because the decision is either yes or no—a mutually exclusive choice—we will use two radio buttons. In Source view, just below the table, insert the text "Provide personal information:" Following the text, drag two radio buttons onto your Source view. Edit the properties for the two radio buttons so they look like this:

```
<asp:RadioButton ID="rbYes" runat="server" AutoPostBack="True"
    Checked="True" GroupName="grpPersonalInfo"
    Text="Yes"
    ToolTip="Click Yes to gather personal information; No to skip that step" />

<asp:RadioButton ID="rbNo" runat="server" AutoPostBack="True"
    GroupName="grpPersonalInfo"
    Text="No"
    ToolTip="Click Yes to gather personal information; No to skip that step" />
```

Each radio button has a unique ID; the first, `rbYes`, and the second, `rbNo`. You'll also notice that both radio buttons have the attribute `runat="server"`. You'll see this attribute on all controls that are evaluated at the server; it's inserted for you automatically.

The text that is displayed next to the `RadioButton` is assigned in the `Text` attribute. You saw earlier in this chapter that a `Button` control performs a postback when it's clicked, by default, but that not all controls do that. In the example, you want the radio buttons to perform a postback, which isn't their normal behavior, so the attribute `AutoPostBack="True"` signals that every time this `RadioButton` is clicked, the page will be sent back to the server for processing. `RadioButtons` are mutually exclusive within their own grouping, meaning that only one button of the group can be checked at a time. The group is established by assigning each radio button a group name, with the `GroupName` property, in this case `grpPersonalInfo`.

Finally, each of these buttons is assigned a tool tip. In this case the tool tip for each button is the same, though that need not be true.

Radio buttons get their name from old-fashioned automobile radios which had mechanical buttons to select the station unlike modern electronic ones that can be used to select more than one station depending on other settings on the radio. These old-fashioned radio buttons physically adjusted the tuner to the desired location. This design was so standardized across all automobiles, that setting and using radio buttons in a car required no more thought than using a water fountain.

For more on this curious idea about self-evident design, we highly recommend the seminal work *The Design of Everyday Things* by Donald A. Norman (Basic Books), which, along with *Don't Make Me Think* by Steve Krug (New Riders) should be required reading for all web application programmers and designers.

The purpose of this radio button group is to allow the user to display or hide the Panel described in the next section. You'll enable that functionality in Chapter 3; for the moment, the radio buttons won't do anything.

Panels

The personal information that you will be gathering will be clustered together within an ASP.NET `Panel` control. Panels give you the opportunity to provide a background color if you choose, or to make the panel itself visible or invisible as a whole. In this example, the visibility of the Panel will be controlled by the radio buttons, although you won't enable that feature until the next chapter.

Begin by dragging a `Panel` control from the Standard section of the Toolbox into the page in Source view, underneath the radio buttons you just added, and giving it the ID and properties as shown here:

```
<asp:Panel ID="pnlPersonalInfo" runat="server"
    BorderWidth="1px" Width="300px" BackColor="beige">
```

What you've done here is pretty self-explanatory: you've defined the width of the panel as 300 pixels, with a one-pixel border. If you hadn't defined the width property, the panel would have sized itself to fit the user's browser automatically. You've also set the background color to beige.

List Selection Controls

Create an HTML table within the panel, like this:

```
<table>
   <tr valign="top">
      <td>
```

The valign property in your first row sets the vertical alignment for all elements within that row to be top-aligned, helping ensure that all of the contents will align properly. Notice that IntelliSense automatically creates the closing tag for each opening tag you type.

Create the first cell by inserting the <td> tag, and type "Areas of Interest." Next, drag a CheckBoxList control into the cell—after the <td> tag and after the text you just added. Switch to Design view; the display should look something like Figure 2-14. Notice the small arrow on the CheckBoxList control; this is a *Smart Tag*. Smart Tags are convenient helpers that provide fast access to essential properties for many controls. Clicking on the arrow opens a small menu.

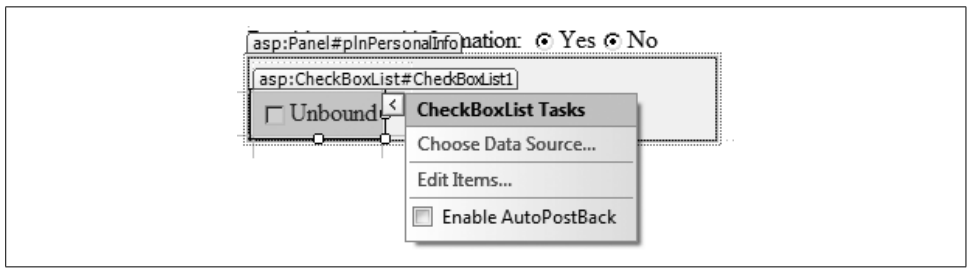


Figure 2-14. *The CheckBoxList you just added shows a Smart Tag to help you set the critical properties of the control.*

Adding items with the Item editor

Click the Smart Tag arrow and select the Edit Items… option to add items to the CheckBoxListcontrol, using the ListItem Collection Editor, shown in Figure 2-15.

With the exception of CheckBox and RadioButton, all the list controls in Table 2-1 hold a collection of ListItem objects. Each ListItem has a Textproperty, which is what is displayed to the user, and a Value property, which you can access programmatically.

This allows you to display text to the user—"Scuba Diving," for example—but when the user selects that option, you'll return a different value to your program—"SC" perhaps, or "4," or whatever value will be meaningful in your application.

The ability to tie a "value" to a "Text property" becomes particularly useful when displaying values retrieved from a database, as we'll see later in this book. You can retrieve, for example, all your vendors, and display them by name, but when one is selected, you can retrieve the vendorID from the value field.

The list is empty when you start, so click the Add button to insert an item. As soon as you add an item, you'll see some familiar-looking properties in the box on the right. Click in the Text field and type "Biking." Notice that you can set the Text and Value properties separately if you choose. If you don't, the Wizard defaults to the

same name for both. You can also set the `Selected` property to True (causing that item to appear as checked when the page is first loaded). For this specific example, add all the items shown in Figure 2-15, set the `Text` and `Value` properties to the same value, and leave all the items unselected and enabled.

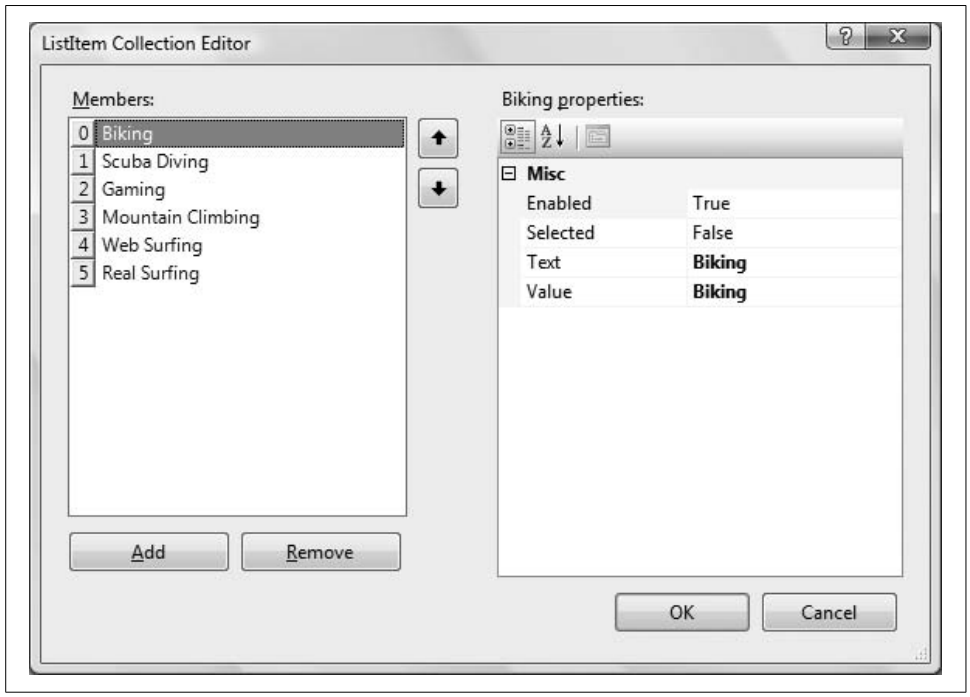


Figure 2-15. *When you select "Edit Items" from the Smart Tag, you'll see the ListItem Collection Editor, where you can enter each item in the list.*

Once you've added all the items to your list, click OK to close the dialog box. Return to the properties window and change the ID property of the `CheckBoxList` to `cblAreas`. Set the `AutoPostBack` property to `True` so that each time a checkbox item is checked or unchecked, the page will be sent back to the server for processing.

Adding items in Source view

Click Source view to see the markup. Press Ctrl-F to bring up the Find dialog, and enter "pnlPersonalInfo" in the Find what box to locate the Panel control. Your IDE should look something like Figure 2-16.

Notice the `CheckBoxList` declaration with its end tag. Between the opening and closing tags are a series of `ListItem` declarations. These are the items you added with the ListItem Collection Editor, and you can change their properties here, if you want.

Now go back to constructing your table—add a second set of cell tags (`<td></td>`) in the same row as the first cell. Press Enter to create a new line, then use the Tab key to indent. Type in the following HTML to form a heading:

```
Age Category
<br />
```

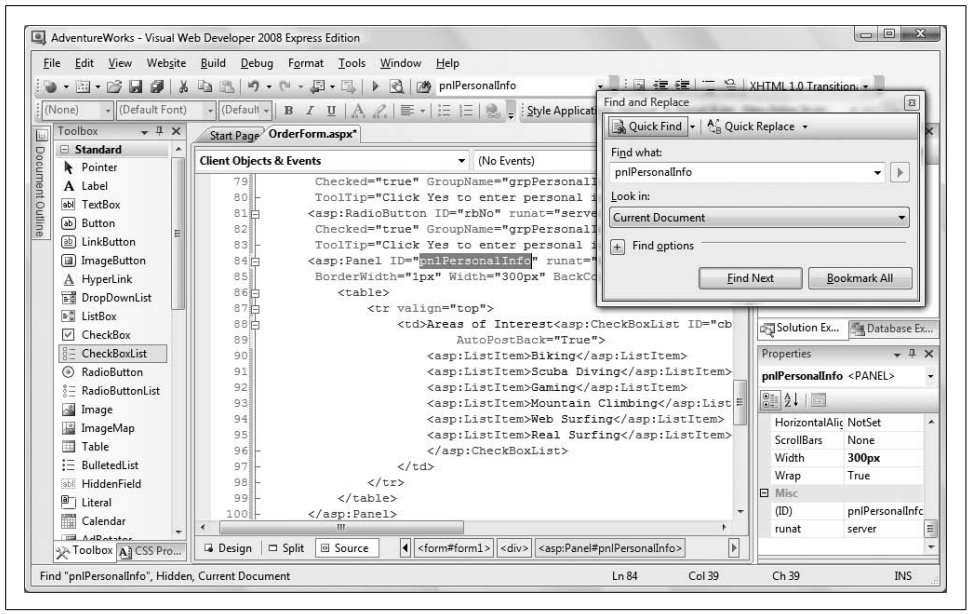


Figure 2-16. *The Source view shows the markup for the Panel control, showing the table and the list items you added.*

Drag a `RadioButtonList` control from the Toolbox onto the source view, directly after the `<br/>` tag. Set the `ID` for the new `RadioButtonList` to `rblAge`, `AutoPostBack` to `True`, and the `Width` to `150`. This time, you'll add `ListItems` to the radio button list by hand. Between the opening tag and the closing tag of the radio button list, type `<asp:ListItem>`. IntelliSense will help you, as shown in Figure 2-17.

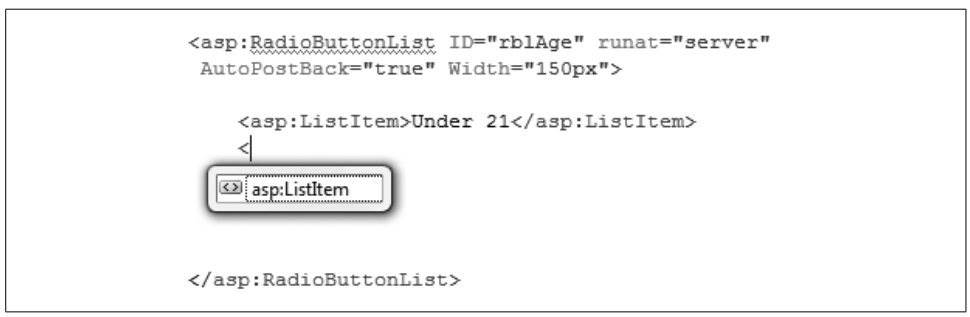


Figure 2-17. *Creating a RadioButtonList by hand. IntelliSense completes the ListItem entry for you.*

Why would you want to add ListItems by hand, rather than use the nifty UI provided by the IDE? Often, it is just a matter of personal preference. Other times, you are already in Source view, and it is just more convenient to stay there. Sometimes, hand editing is the only way to get the markup exactly as you need it to be. And often times, it is just faster by hand because you can copy and paste similar lines of code and change only one or two attributes.

You have now added one list item, `Under21`, and opened the angle bracket for a second `ListItem`. `IntelliSense` knows the only possible control that can go in this location is an ASP.NET `ListItem`, and so it offers that option to you. You can click on the `ListItem` offered by IntelliSense to explicitly select it, or simply press tab to accept it.

When you enter the closing angle bracket (>), the IDE will immediately create a closing tag for you:

```
<asp:ListItem></asp:ListItem>
```

You need only put your new value between the tags. Thus, you can quickly build the contents of your radio button list. Add the rest of the `ListItems` now, so the Source view looks like this:

```
<asp:RadioButtonList ID="rblAge" runat="server"
AutoPostBack="True" Width="150px">
   <asp:ListItem>Under 21</asp:ListItem>
   <asp:ListItem>21 to 30</asp:ListItem>
   <asp:ListItem>31 to 50</asp:ListItem>
   <asp:ListItem>Over 50</asp:ListItem>
</asp:RadioButtonList>
```

Placing the text inside the tags like this is equivalent to setting the `Text` property of the `ListItem`, which is what you did in the ListItem editor. Now, switch back to Design view. You should see something like Figure 2-18.

In the layout table at the top of the page, you gathered the user's name and address. You'll remember that we planned to use a drop-down list for the state field. It would be convenient to display the full name of the state while setting the corresponding `value` property to the two-letter postal abbreviation.

Go back up to the layout table and drag a `DropDownList` control into the cell for State, and name the control `ddlState`. At this point, you have two options for setting the text and value property: you can either use the `ListItem` Collection Editor in Design view, or you can fill in the list items by hand in Source view. Add the following four items to the `DropDownList` now, so the Source view looks like this:

```
<asp:DropDownList ID="ddlState" runat="server">
   <asp:ListItem Value="AL">Alabama</asp:ListItem>
   <asp:ListItem Value="AK">Alaska</asp:ListItem>
   <asp:ListItem Value="CA">California</asp:ListItem>
   <asp:ListItem Value="CT">Connecticut</asp:ListItem>
```

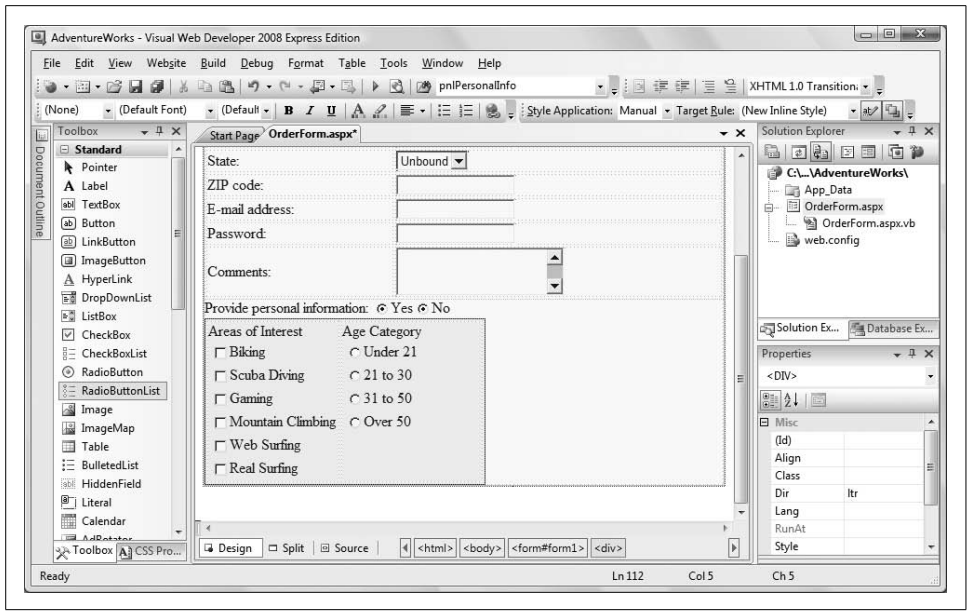


Figure 2-18. *Design view with personal information controls in place in a Panel control.*

In a production environment, you would probably retrieve the text and the value from columns in a database table.

More Selection Controls

Again referring to Figure 2-9, you need to create two more list selection controls: one for the product category, and one for the subcategory. Begin by inserting a new layout table just as you did previously, but below `pnlPersonalInfo`. Give the table one row and four columns. In the first cell, type "Product Category:"; in the third cell, type "Subcategory:".

Drag a `DropDownList` into the second cell and a `ListBox` into the fourth cell. Using the Properties window, change the IDs of these two controls, and set the following properties.

Property	DropDownList	ListBox
`ID`	`ddlCategory`	`lbSubcategory`
`SelectionMode`		`Single`
`ToolTip`	Select a category	Select a subcategory
`AutoPostBack`	`True`	`True`

Use the Smart Tag and the `ListItem` Collection Editor, as you did in Figure 2-15, to enter the following `ListItems` for each control.

DropDownList	ListBox
Bikes	Brakes
Components	Handlebars
Clothing	Chains
Accessories	Cranks
Scuba	Bottom Brackets
Parasailing	Tires
	Wheels
	Seats
	Derailleurs

One final layout tweak: go to Source view, find the HTML `<table>` currently under construction, then add a `valign` attribute to the row tag, `<tr valign="top">` to top align all the elements in the row to the top of the table.

Look at this section of the page in Design view. Figure 2-19 shows how the product table should appear at this point.

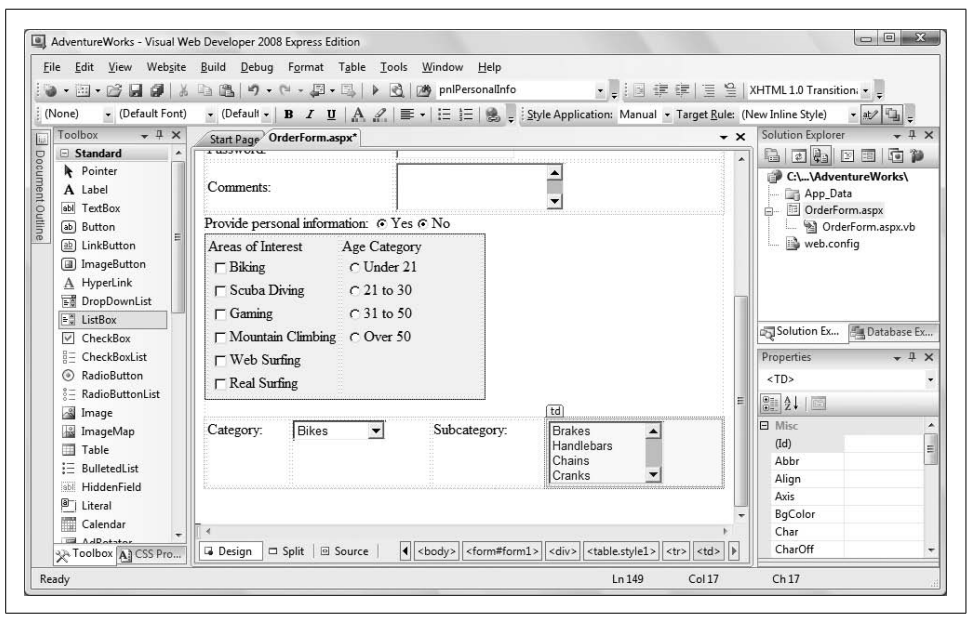


Figure 2-19. *Design view of the product category lists after adding the last set of selection controls.*

Using Selections to Display Text

So far, you've created controls that provide choices to the user, but you haven't seen how your page knows what items the user selected, or how to do anything with them. The answer lies in the properties of the selection controls. The `DropDownList` control, and all of the other list controls back in Table 2-1 (except `CheckBox` and `RadioButton`) provide three properties for retrieving selections:

`SelectedIndex`
: Returns or sets the index of the selected item. The index is a number (an integer) indicating which `ListItem` you chose. The index always starts with zero, so the first item in the list is index 0, the second item is index 1, and so on. If more than one item is selected, it returns the lowest index of all the selected items, or –1 if nothing is selected. If you set this property, it deselects all the selected items except the one with the specified index.

`SelectedItem`
: Returns the selected `ListItem`. If more than one item is selected, it returns the one with the lowest index. If nothing is selected, it returns `Nothing` (`null` in C#).

`SelectedValue`
: Returns or sets the `Value` property of the selected item as a `String`. If more than one item is selected, it returns the `Value` of the selected item with the lowest index. If none of the items in the list control matches the `SelectedValue`, an error message is sent.

When a user submits this form, you want to provide feedback regarding the selected product category, subcategory, and the mailing address to which the purchase will be sent. A summary table is the solution.

To demonstrate three different ways of displaying text:

- You'll display the chosen Category in a label.
- You'll display the chosen Subcategory in a read-only text box.
- Finally, you'll display the Mailing Address by modifying the *inner HTML* of the table cell itself. (Inner HTML is the content between the opening and closing tags.)

To begin, you need to create the layout table with three rows and two columns underneath the existing controls. You can type it directly into the source, or use the Insert Table Wizard we showed you earlier. Whichever method you choose, the table should end up looking like this in Source view:

```
Summary<br />
<table>
   <tr valign="top">
      <td>
         Category:
      </td>
```

```
        <td>
            <asp:Label ID="CategoryLabel" runat="server" Text="" />
        </td>
    </tr>
    <tr valign="top">
        <td>
            SubCategory:
        </td>
        <td>
            <asp:TextBox ID="SubCategoryTextBox" runat="server"
            ReadOnly="true" />
        </td>
    </tr>
    <tr valign="top">
        <td >
            Mailing Address:
        </td>
        <td id="tdAddress" runat="server" style="width:200px">
        </td>
    </tr>
</table>
```

This is a fairly straightforward HTML table. The left column has "Category" in the first row, "Subcategory" in the second, and "Mailing Address" in the third row.

In the right column, place a `Label` control in the first row with an ID of `CategoryLabel`, but with its `Text` property set to an empty string. You'll fill that at runtime. In the second row, insert a `TextBox` control that has its `ReadOnly` property set to true, so that the user cannot type into the text box (you're using it for display only).

The third row's second column is a bit unusual; the `<td>` itself has an `id` and a `runat` attribute, making this an HTML server control. This allows you to modify the cell itself programmatically.

If you want to populate the summary table, you'll need to perform a postback to evaluate and process the code (as discussed in Chapter 1). To do that, you'll need a Submit button. In Design view, drag a `Button` control onto the page just below the summary table. Change its `ID` to `btnSubmit`, and its text to "Submit."

The Submit button's `Click` event handler will populate the `Text` properties of the `Label` and of the `TextBox`, and will set the inner HTML of the third row's second column, the one which is an HTML server control. Double-click the Submit button from Design view, and you'll be automatically taken to the `Click` event handler in the code behind file. Add the following code to wire up the functionality:

```
Protected Sub btnSubmit_Click( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnSubmit.Click
```

```
    CategoryLabel.Text = ddlCategory.SelectedItem.Text

    SubCategoryTextBox.Text = lbSubCategory.SelectedItem.Text

    Dim strMailingAddress As String
    strMailingAddress = txtName.Text + "<br/>" + _
                        txtAddress.Text + "<br/>" + _
                        txtCity.Text + ", " + _
                        ddlState.SelectedValue + " " + _
                        txtZip.Text
    tdAddress.InnerHtml = strMailingAddress
End Sub
```

Let's take a closer look at this code. The event handler retrieves the selected item from the drop-down list for Categories (`ddlCategory`) and asks it for its `Text` property, which it then assigns to the `Text` property of the `CategoryLabel`:

```
CategoryLabel.Text = ddlCategory.SelectedItem.Text
```

Similarly, the text is retrieved from the `SelectedItem` property of the `ListBox` that holds the Subcategory, and that text is assigned to the `Text` property of the read-only `TextBox`:

```
SubCategoryTextBox.Text = lbSubCategory.SelectedItem.Text
```

Finally, and this is a bit tricky, the text of the various address fields are retrieved (including the selected value from the state drop-down), joined into a single text string, and assigned to the local variable `strMailingAddress`. That value is then assigned to the `InnerHtml` property of `tdAddress`. This is, you'll remember, the ID assigned to the second `<td>` tag of the third row. The net result is that the cell is filled with the address string:

```
Dim strMailingAddress As String
strMailingAddress = txtName.Text + "<br/>" + _
                    txtAddress.Text + "<br/>" + _
                    txtCity.Text + ", " + _
                    ddlState.SelectedValue + " " + _
                    txtZip.Text
tdAddress.InnerHtml = strMailingAddress
```

If you run this code and click on the Submit button without first selecting a SubCategory, you will get an error that says "NullReference-Exception was unhandled by user code—Object reference not set to an instance of an object". (You will learn how to debug this in Chapter 8.) This error is caused by the fact that the `SelectedItem` property returns `Nothing` if nothing is selected, and our code is trying to get the `Text` property of `Nothing`, which does not exist. To fix this, set the `Checked` property of one of the `ListItems` to `True`, as in:

```
<asp:ListItem Selected="True">Brakes</asp:ListItem>
```

The final result is shown in Figure 2-20. Run your application and try it out. When you enter text in the text boxes and make selections in the category fields, and then click the Submit button, the Summary table updates with the text you've entered.

VB CHEAT SHEET

Variables and Strings

In the first two controls in the Summary section of the example, you simply assigned the `Text` property of a `SelectedItem` property to the `Text` property of another control; that's easy enough. But for the third control, you took the `Text` properties of several controls, joined them together, and assigned them as a whole.

The trick to this is using a *variable*. Simply put, a variable is like a bucket in your code, which can be used to hold a value. You can retrieve the value later, change it, or replace it with another value. You don't need to worry about what the value is when you're writing your code; you just need to know the name of the variable. In this example, you're using a variable named `strMailingAddress` to hold the text of the user's address.

In VB, you create a new variable using the `Dim` statement, followed by the name you want to give the variable:

```
Dim strMailingAddress As String
```

You also need to give the variable a *type*, which tells the compiler what kind of data it can expect to find in the variable. In this example, the variable consists of text, and in VB, a sequence of text is called a *string*. You use the keyword `As` to declare a variable named `strMailingAddress`, of type `String`. The important thing to know about strings is that all literal string values are surrounded by double quotes (`""`).

One of the useful things about strings is that you can take two strings and put them together into a single, longer string. This is called *concatenation*, and it's very easy to do in VB; you just use the `+` operator. Look at this bit of code:

```
txtName.Text + "<br/>" + txtAddress.Text + "<br/>"
```

All this does is take the string in `txtName.Text`, add to it the string that represents a line break in HTML (`<br/>`), add the string from `txtAddress.Text`, and then add another line break. All of that gets assigned to the variable `strMailingAddress`, which in turn gets assigned to the inner HTML of the `<td>` element.

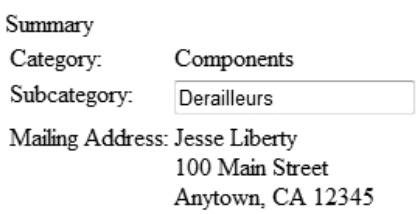


Figure 2-20. *This is what the Summary Table of the page looks like after the user has entered values in the top part of the page.*

Images

Images are an important addition to any web site. An image can be a photograph, a drawing, a logo—any graphic.

ASP.NET provides several controls to work with images:

- An `Image` control is used to display an image. We will demonstrate this momentarily.
- An `ImageButton` is used to create an image that can be clicked, thus giving it the behavior of a normal button.
- An `ImageMap` control provides an image with multiple clickable hotspots. Each of the hotspots behaves like a separate hyperlink.

Now insert an `Image` control into the form. To do so, insert some room below the Summary table and the Submit button by hitting the Return key a few times, and then drag a `CheckBox` and an `Image` control onto the form. In the Properties window, set the `ID` of the `CheckBox` to `cbDisplayPhoto`, and be sure to set `AutoPostBack` to `True`, `Checked` to `True`, and `Text` to "Show product photo?" Also, set the `TextAlign` property to `Left`.

Set the ID for the image to `imgPhoto` and the `ImageURL` to "Dan at Vernal Pool.jpg". We've provided an image for you to use in this example. You can download that image with the code for the book from *www.LibertyAssociates.com*. Once you have the file, simply drag and drop it onto the AdventureWorks folder in Solution Explorer. You'll see the image file appear in the file tree alongside your other solution files. You can also use any image file you have handy.

The `Image` control has only three essential properties: the ID (so that you can address the control programmatically), the ubiquitous `runat="server"`, and the `ImageUrl` that identifies the location of the image. Because you put this image in the base directory of the application, you do not need a pathname, only the name of the file itself.

On this page, the `CheckBox` control offers the user the opportunity to make the image visible or not. It has its `AutoPostBack` property set to true to force a postback every time the `Checked` property changes. To make use of this, of course, you must write an event handler for the `CheckedChanged` event. Double-click the `CheckBox` to create an event handler for `CheckChanged`, and add the following highlighted line of code:

```
Protected Sub cbDisplayPhoto_CheckedChanged( _
        ByVal sender As Object, _
        ByVal e As System.EventArgs) _
    Handles cbDisplayPhoto.CheckedChanged
    imgPhoto.Visible = cbDisplayPhoto.Checked
End Sub
```

This event handler changes the `Visible` property of the `Image`. When the property is set to false, the image isn't rendered. Go ahead and try it out. You'll see that when you uncheck the box, the page posts back, and the image vanishes.

Links

Hyperlinks provide immediate redirection to another page in the application or to a location elsewhere on the Internet without posting back to the server. We'll use a `HyperLink` control to provide a link to Jesse's home page, serving here very much the same function as an `<a>` tag would do in HTML.

Add the text, "For help, contact", and then drag a `HyperLink` control onto the bottom of your form. Set the ID to `hypContact`, its `NavigateURL` to *http://www.JesseLiberty.com*, and its text (which will become its inner HTML) to "Jesse Liberty's Portal". Finally, set the `Target` to `"_blank"` (we'll explain this property in just a moment).

Switch to Source view, and you'll see that the markup produced looks something like this:

```
<br />
For help, contact
<asp:HyperLink ID="hypContact" runat="server"
    NavigateUrl="http://www.JesseLiberty.com"
    Target="_blank">
    Jesse Liberty's Portal
</asp:HyperLink>
<br />
```

This last property, `Target`, specifies in which window or frame the new page will open. You can specify a window by name, or use one of the special values listed in Table 2-2.

Table 2-2. *Special values of the Target attribute*

Target value	Description
`_blank`	Renders the content in a new unnamed window without frames.
`_new`	Not documented, but behaves the same as `_blank`.
`_parent`	Renders the content in the parent window or frameset of the window or frame with the hyperlink. If the child container is a window or top-level frame, it behaves the same as `_self`.
`_self`	Renders the content in the current frame or window with focus. This is the default value.
`_top`	Renders the content in the current full window without frames.

By setting the value of `Target` to `_blank`, clicking on the link instructs the target page to open in a new browser window.

One significant advantage of using this control over an `<a>` tag is the ID and `runat="server"` properties, which allow you to address the control programmatically. For example, you could set the `NavigateUrl` to a different location from within your code, based on conditions established while the program is running.

LinkButtons

Remember that a hyperlink redirects immediately and does not post back first; thus, there is no server-side processing possible when the user clicks the link. If you want the *appearance* of a hyperlink, but need to perform server-side processing before departing for the new page (e.g., to save data to a database) then use a `LinkButton` control. The `LinkButton` behaves like a `Button` but looks like a `HyperLink`, and the behavior is to post back to the server, do its work and then redirect the user to a new location. You already have a `HyperLink` control on this page, so you won't add a `LinkButton`.

In order to accomplish the redirection to the new page, you'd need to use the `Response.Redirect` method, as shown in the following click event handler:

```
Protected Sub MyLinkButton_Click( _
       ByVal sender As Object, _
       ByVal e As System.EventArgs) _
       Handles MyLinkButton.Click
             'Save data to db
             Response.Redirect("http://www.JesseLiberty.com")
End Sub
```

Source Code

For your convenience (in case you are away from your computer) the complete source code is shown below. The OrderForm markup is shown in Example 2-3. The code behind file is shown directly after in Example 2-4.

Example 2-3. *OrderForm.aspx*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="OrderForm.aspx.vb"
    Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>AdventureWorks</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1>
            AdventureWorks Order Form</h1>
        <table >
            <tr>
                <td>Customer Name:</td>
                <td>
                    <asp:TextBox ID="txtName" runat="server"></asp:TextBox>
                </td>
```

Example 2-3. *OrderForm.aspx (continued)*

```
        </tr>
        <tr>
            <td>Address:</td>
            <td>
                <asp:TextBox ID="txtAddress" runat="server"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>City:</td>
            <td>
                <asp:TextBox ID="txtCity" runat="server"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>State:</td>
            <td>
                <asp:DropDownList ID="ddlState" runat="server">
                    <asp:ListItem Value="AL">Alabama</asp:ListItem>
                    <asp:ListItem Value="AK">Alaska</asp:ListItem>
                    <asp:ListItem Value="CA">California</asp:ListItem>
                    <asp:ListItem Value="CT">Connecticut</asp:ListItem>
                </asp:DropDownList>
            </td>
        </tr>
        <tr>
            <td>Zip:</td>
            <td>
                <asp:TextBox ID="txtZip" runat="server"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>E-mail:</td>
            <td>
                <asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>Password:</td>
            <td>
                <asp:TextBox ID="txtPassword" runat="server"
                    TextMode="Password"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>Comment:</td>
            <td>
                <asp:TextBox ID="txtComment" runat="server" Rows="3"
                    TextMode="MultiLine"></asp:TextBox>
            </td>
        </tr>
    </table>
    Provide Personal Information:
```

Example 2-3. *OrderForm.aspx (continued)*

```
    <asp:RadioButton ID="rbYes" runat="server" AutoPostBack=true
        Checked="true" GroupName="grpPersonalInfo"
        Text="Yes"
        ToolTip="Click Yes to gather personal information; No to skip that step" />
    <asp:RadioButton ID="rbNo" runat="server" AutoPostBack=true
        GroupName="grpPersonalInfo"
        Text="No"
        ToolTip="Click Yes to gather personal information; No to skip that step" />
    <asp:Panel ID="pnlPersonalInfo" runat="server"
        BorderWidth="1px" Width="300px" BackColor="Beige">
        <table>
            <tr valign="top">
                <td>
                    Areas of Interest
                    <asp:CheckBoxList ID="cblAreas" runat="server"
                        AutoPostBack="True">
                        <asp:ListItem>Biking</asp:ListItem>
                        <asp:ListItem>Scuba Diving</asp:ListItem>
                        <asp:ListItem>Gaming</asp:ListItem>
                        <asp:ListItem>Mountain Climbing</asp:ListItem>
                        <asp:ListItem>Web Surfing</asp:ListItem>
                        <asp:ListItem>Real Surfing</asp:ListItem>
                    </asp:CheckBoxList>
                </td>
                <td>
                    Age Category
                    <br />
                    <asp:RadioButtonList ID="rblAge" runat="server"
                        AutoPostBack="true" Width="150px">
                        <asp:ListItem>Under 21</asp:ListItem>
                        <asp:ListItem>21 to 30</asp:ListItem>
                        <asp:ListItem>31 to 50</asp:ListItem>
                        <asp:ListItem>Over 50</asp:ListItem>
                    </asp:RadioButtonList>
                </td>
            </tr>
        </table>
    </asp:Panel>
    <table>
        <tr valign="top">
            <td>
                Product Category
            </td>
            <td>
                <asp:DropDownList ID="ddlCategory" runat="server"
                    AutoPostBack="true"
                    ToolTip="Select a category">
                    <asp:ListItem>Bikes</asp:ListItem>
                    <asp:ListItem>Components</asp:ListItem>
                    <asp:ListItem>Clothing</asp:ListItem>
                    <asp:ListItem>Accessories</asp:ListItem>
```

Example 2-3. *OrderForm.aspx (continued)*

```
                    <asp:ListItem>Scuba</asp:ListItem>
                    <asp:ListItem>Parasailing</asp:ListItem>
                </asp:DropDownList>
            </td>
            <td>
                SubCategory
            </td>
            <td>
                <asp:ListBox ID="lbSubCategory" runat="server"
                    AutoPostBack="true"
                    SelectionMode="Single"
                    ToolTip="Select a sub-category">
                    <asp:ListItem Selected="True">Brakes</asp:ListItem>
                    <asp:ListItem>Handlebars</asp:ListItem>
                    <asp:ListItem>Chains</asp:ListItem>
                    <asp:ListItem>Cranks</asp:ListItem>
                    <asp:ListItem>Bottom Brackets</asp:ListItem>
                    <asp:ListItem>Tires</asp:ListItem>
                    <asp:ListItem>Wheels</asp:ListItem>
                    <asp:ListItem>Seats</asp:ListItem>
                    <asp:ListItem>Derailleurs</asp:ListItem>
                </asp:ListBox>
            </td>
        </tr>
    </table>
    <table border="1">
        <tr valign="top">
            <td>
                Category:
            </td>
            <td>
                <asp:Label ID="CategoryLabel" runat="server" Text="" />
            </td>
        </tr>
        <tr valign="top">
            <td>
                SubCategory:
            </td>
            <td>
                <asp:TextBox ID="SubCategoryTextBox" runat="server"
                    ReadOnly="true" />
            </td>
        </tr>
        <tr valign="top">
            <td>
                Mailing Address:
            </td>
            <td id="tdAddress" runat="server" style="width: 200px">
            </td>
        </tr>
    </table>
```

Example 2-3. *OrderForm.aspx (continued)*

```
        <asp:Button ID="btnSubmit" runat="server" Text="Submit" />
        <br />
        <br />
        <asp:CheckBox ID="cbDisplayPhoto" runat="server" AutoPostBack="True"
            Checked="True" Text="Show product photo?" TextAlign="Left" />
        <br />
        <asp:Image ID="imgPhoto" runat="server" ImageUrl="Dan at Vernal Pool.jpg" />
        <br />
        <br />
        <asp:HyperLink ID="hypContact" runat="server"
            NavigateUrl="http://www.JesseLiberty.com" Target="_blank">
                Jesse Liberty's Portal
        </asp:HyperLink>
    </div>
    </form>
</body>
</html>
```

Example 2-4. *OrderForm.aspx.vb*

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub btnSubmit_Click( _
        ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles btnSubmit.Click

        CategoryLabel.Text = ddlCategory.SelectedItem.Text
        SubCategoryTextBox.Text = lbSubCategory.SelectedItem.Text
        Dim strMailingAddress As String
        strMailingAddress = txtName.Text + "<br/>" + _
                            txtAddress.Text + "<br/>" + _
                            txtCity.Text + ", " + _
                            ddlState.SelectedValue + " " + _
                            txtZip.Text
        tdAddress.InnerHtml = strMailingAddress
    End Sub

    Protected Sub cbDisplayPhoto_CheckedChanged( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles cbDisplayPhoto.CheckedChanged
        imgPhoto.Visible = cbDisplayPhoto.Checked
    End Sub

End Class
```

Summary

- A *postback* occurs when an event happens on your page that causes the page to return to the server, handle the events, and then send the same page back to the browser. The contents of the page may have changed, but the page object itself is the same.
- Postbacks can be either synchronous, in which case the entire page is returned to the server, or, with AJAX, asynchronous, in which case only part of the page is returned to the server.
- A *control* is a tool that lets your web page take an action. It could be as simple as displaying some text, or as complicated as interacting with a database. Most controls have some visual representation that the user sees, although not all do.
- Placing a control in your web page is as simple as dragging it from the Toolbox onto your page; the IDE inserts the appropriate markup for you. Controls all come with at least a few properties and methods, which you can use to customize their appearance and behavior, respectively.
- Every control has a unique identifier, its `ID` property. The IDE assigns a default ID automatically, but you can (and usually should) rename them to be more meaningful.
- Almost every control has associated events, as well as properties. You can access these by clicking the Events button in the Properties window.
- You can create tables by hand in Source view, or you can use the Insert Table Wizard by selecting Layout → Insert Table in Design view.
- The `TextBox` control is a relatively simply control that allows the user to enter text that you can retrieve later. You can change the `TextMode` property of a `TextBox` to create single-line entry fields, multiline fields, or to hide the text for a password field.
- ASP.NET has a number of *selection controls*, including the `ListBox`, `DropDownList`, `RadioButton`, `RadioButtonList`, `CheckBox`, and `CheckBoxList`, which display various options for the user to choose from. You decide which control to use based on its appearance, and whether you want the user to be able to make only one selection from within a list or multiple selections.
- If the `AutoPostBack` property of a control is set to `True`, the page is posted back to the server whenever that control's value changes.
- Radio buttons can be assigned to a group, by setting each button's `GroupName` property. That ensures that only one button in a group can be selected at a time. You can also use a `RadioButtonList` to accomplish the same thing.
- You can use a `Panel` control to group other controls together, and also to make the content in the panel visible or invisible as a group.

- Many controls have a Smart Tag, which is a small menu that provides quick access to the most common tasks for that control. In the case of selection controls, the Smart Tag lets you access the `ListItem` editor.
- The selection controls each contain a collection of `ListItem` objects, which you use to offer the user choices to select from. The `Value` property of the `ListItem` can be different from the `Text` property that you display to the user, and you can retrieve the selected value for later use. The ListItem Collection Editor makes it easy to add `ListItems`, but you can also add them by hand in Source view.
- There are three properties that let you retrieve the items that users select from a selection control: `SelectedIndex` gets the index of the selected item, `SelectedItem` gets the `Text` property, and `SelectedValue` gets the `Value` property. You can use these values to display the selected item in another control, or to use it in other ways.
- There are several ways to display dynamically generated read-only text in your page: among others, you can set the property of a `Label` control, you can use a read-only `TextBox` control, or you can set the inner HTML of an HTML element.
- You use an `Image` control to display an image or graphic. The `ImageButton` control displays an image, and acts like a button. An `ImageMap` control displays an image that has multiple areas that the user can click, each acting like a hyperlink or button.
- The `Visible` property of a control determines whether that control is rendered on the page. You can change the value of this property programmatically, and cause the control to appear or disappear with a postback.
- A `HyperLink` control works like an HTML anchor `<a>` tag. You can set the `NavigateURL` property and the text of the hyperlink separately. You can also specify if the link's target will open in a new page or a new frame with the `Target property`.

You've got a lot of things in your toy box now, and you can do a lot more than just the label and button from *Hello World* in Chapter 1. In fact, in this chapter, you've just seen the more common controls—there are many others out there that are more specialized, such as the `Calendar` and `AdRotator` controls, and that's not even leaving the General tab of the Toolbox. Feel free to experiment with them. Now that you have a base to work from, in the next chapter, we'll show you how you can use AJAX to do some clever things with the plain-vanilla controls you just learned about.

Quiz

1. What is a postback?
2. What are the two types of postbacks in ASP.NET, and what is the difference between them?
3. What property is found on every control?
4. What control would you use to have the user enter a password, but keep the text hidden?
5. What control would you use if you have a list of 20 items, and the user can select as many as they want?
6. How do you make single radio buttons mutually exclusive?
7. What can you use a `Panel` control for?
8. What does the `SelectedItem` property retrieve?
9. How do you include a control on the page, but not render it?
10. What do you do to make the target of a `HyperLink` control open in a new window?

Exercises

Exercise 2-1. Now that you've played with *Hello World*, you're going to make a change to the Postbacks example from this chapter, so you can see how flexible the `UpdatePanel` control is. Open the Postbacks web site (similar to how you opened *Hello World* in the previous exercise). In Design view, drag another `UpdatePanel` control inside the first one, after the button. Click inside the new `UpdatePanel` and type the text "Another partial-page update:". Drag another `Label` control inside the new `UpdatePanel`. In the Properties window, set the label's name to `lblOtherPartialUpdate`, and set its width to `200px`. (Note that you can't give this label the same name as the other label—or any other control on the page—or you'll get an error.) Now, add another `Button` to the new `UpdatePanel`, under the label, set its name to `btnOtherPartialUpdate`, and change the text to "Another Partial Update".

You need the event handler for your new button, so double-click it, and you'll be taken to the code-behind file. You'll see the event handlers for the two existing buttons already there, and the skeleton for the new event handler. Add the following line of code to this new event handler so it will update with the current time, like the other two buttons do:

```
lblOtherPartialUpdate.Text = DateTime.Now
```

Run your application, and click the buttons. If all went well, you'll see that each label updates independently from the others, and that the two buttons in the update panels don't cause any page flicker. Your page should look something like Figure 2-21.

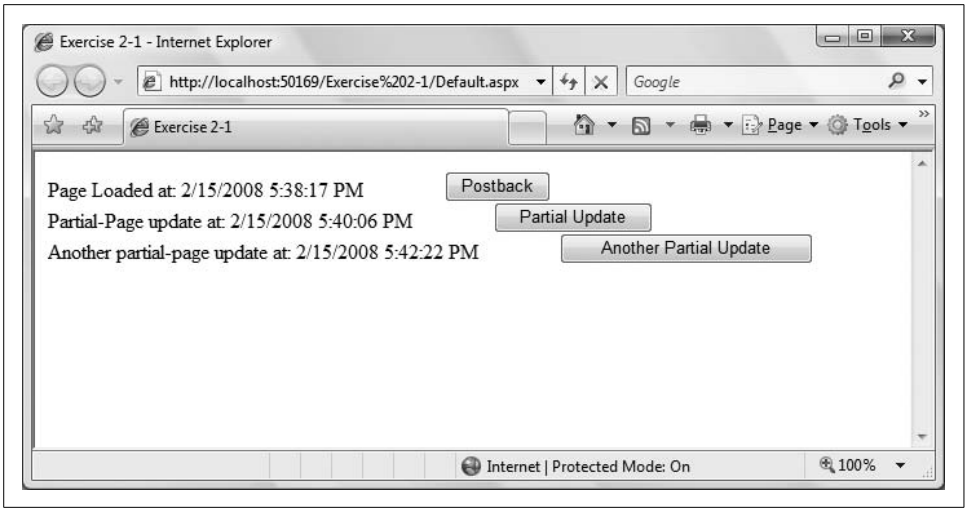


Figure 2-21. *Your goal for Exercise 2-1. Each label should update independently of the others.*

Exercise 2-2. When you're creating a web page, often knowing which controls to use is a bigger challenge than using the controls properly. That's what you're going to practice here. Imagine a page for a busy ice cream shop that lets you preorder an ice cream cone so it will be ready for you when you get to the counter. The page should have three controls. The first control asks users to select the type of ice cream from the following list: Vanilla, Chocolate, Strawberry, Mint, Butter Pecan, Coffee, Pistachio, Coconut, Bubble Gum, and Cotton Candy. Only one type of ice cream is allowed per order. The second control asks the user to select the toppings they want: chocolate sprinkles, rainbow sprinkles, hot fudge, caramel, cookie dough, Oreo cookies, pretzel bits, walnuts, coffee beans, or crushed candy bars. It's a gourmet ice cream shop, so customers can have as many toppings as they like. The third control asks users to choose a cone or a dish. Obviously, only one is allowed. Make sure to include a way for users to submit their order.

Exercise 2-3. Now that you've made a working page with different controls, it's time to try retrieving a value. Create a page with a simple `TextBox` that asks the user to enter his or her password. The password should be disguised as the user types it. Then, with shocking disregard for security, use a label control to repeat the user's password back to him. The page should look something like Figure 2-22.

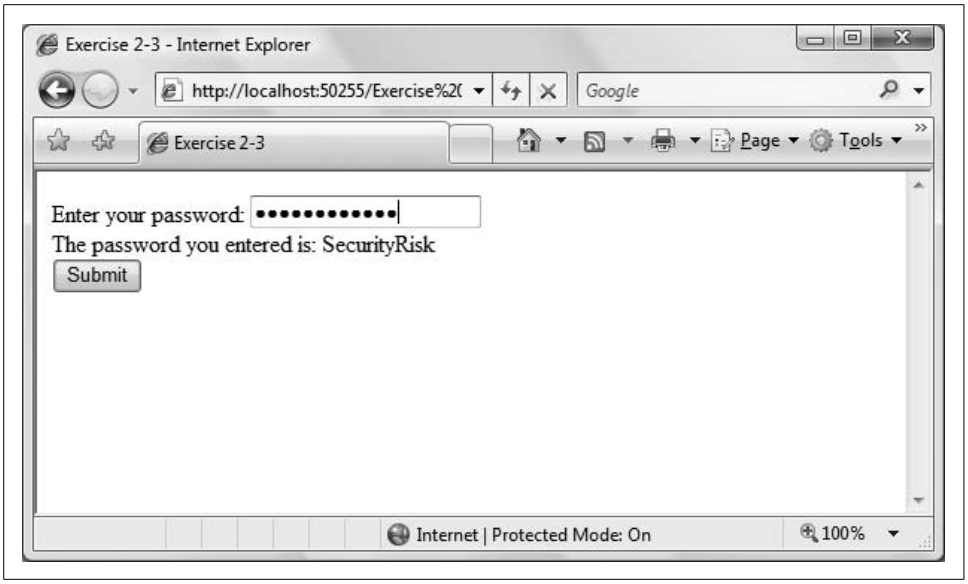


Figure 2-22. *Your goal for Exercise 2-3.*

Exercise 2-4. Now you're ready to try a slightly more complicated example. Create a drop-down list that presents a number of fine books from your authors; call it `ddlBookList`. This time, the `Text` and `Value` properties of the `ListItems` in the drop-down should be different, according to the following table.

Text	Value
Programming ASP.NET	00916X
Programming C#	006993
Programming Visual Basic.NET	004385
Learning C# 2008	102097

These values are part of each book's ISBN, and were you a bookstore or a warehouse, you would probably use a database with these numbers to help keep track of the books you have in stock. In this case, though, you'll just show the user what they selected, including the value. Add two labels to show the results, as shown in Figure 2-23.

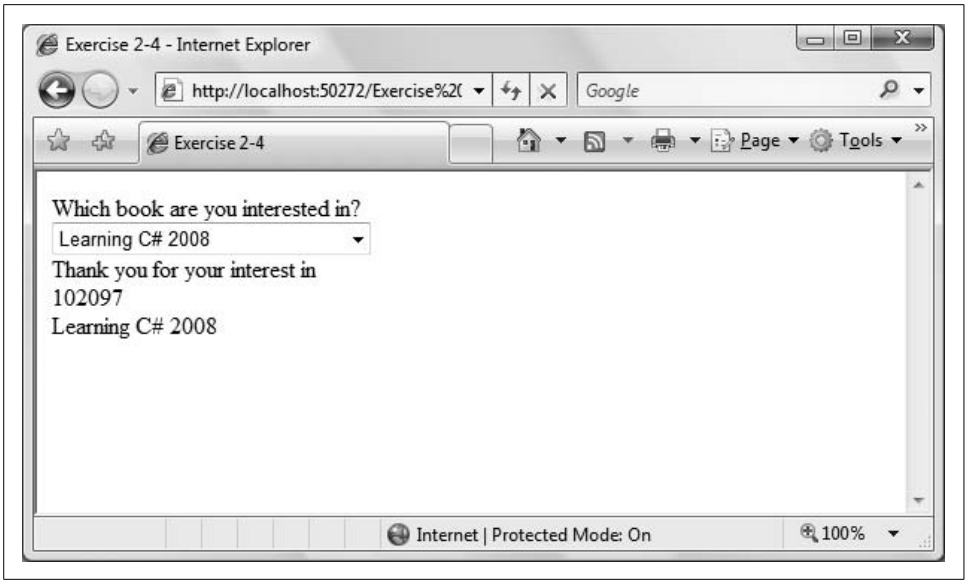


Figure 2-23. *Your goal for Exercise 2-4.*

CHAPTER 3

Snappier Web Sites with AJAX

AJAX has revolutionized ASP.NET, and from this moment forward, most ASP.NET applications will routinely integrate AJAX controls. AJAX moves ASP.NET applications from being 99% server-side code to offering the option for a great deal of the processing to happen on the user's browser. The net effect is a tremendous increase in both real and perceived performance of ASP.NET applications.

To demonstrate how much more dynamic and responsive AJAX is, you'll rewrite the order form from Chapter 2, applying AJAX techniques. You'll enhance the site by adding a watermark to user entry fields. A watermark is a bit of text that appears in the text field itself, but disappears as soon as the user starts typing. It serves as an elegant prompt to the user. You will also create a pop-up panel to hide controls until the user needs them, and you'll add a collapsible text field to display product information in a very space-efficient manner.

Take a Walk on the Client Side

While server-based web applications have wonderful advantages, they have the obvious disadvantage that any time you want to run any code (or retrieve any data) you must endure the cost of a "round trip" from the browser to the server and back, and the page must be redrawn from scratch. Round trips can be slow (though the Internet is getting faster all the time), and redrawing the page causes a noticeable flicker.

AJAX (which more accurately should be spelled AJX, but that's harder to pronounce) is an acronym for Asynchronous JavaScript and XML—that is, it is a technique for combining well established (some might say *old*) Internet technology in new ways to greatly enhance the performance of web applications. AJAX enabled applications are very hot—they outperform server-based applications in ways that would make your jaw drop.

Microsoft, realizing this was not a technology they could ignore, and having learned the lesson that they must leave open standards open, chose to take this very good idea and make it much *much* better, without making it proprietary.

AJAX Doesn't Exist

There really isn't any such thing as AJAX. It isn't a product or a standard; it isn't even a technology. It is just a way to refer to a set of existing technologies used together in new ways to do cool things.

The first use of the term as an acronym for "Asynchronous JavaScript and XML" was by Jesse James Garrett in February 2005. Garrett thought of the term while in the shower (if you must know), when he realized the need for a shorthand term to represent the suite of technologies he was proposing to a client (who, we are assured, was not in the shower with him).

On the other hand, the first use of the term at all may have been nearly 3000 years earlier, by Homer, who wrote about Ajax the Great (and also Ajax the Lesser) in the *Iliad* (Book 7, 181–312). Ajax the Great was the tallest and strongest of the Achaeans, and second only to Achilles in skill as a warrior. It isn't clear if the tale of AJAX-The-Technology will be told 3,000 years from today (or even 3,000 days), though we're pretty certain there is a parallel between the Trojan War and the desktop wars, but that is for another book.

According to Garrett, "AJAX…is really several technologies, each flourishing in its own right, coming together in powerful new ways." AJAX incorporates:

- *Standards-based presentation* using XHTML and CSS, with dynamic display and interaction using the *Document Object Model* (DOM). This allows AJAX, through JavaScript code, to directly manipulate any element on the page.
- Data interchange and manipulation using *XML* and *XSLT*, a nonproprietary and platform independent way of working with data, allowing AJAX to work on any platform using industry standard technology.
- Asynchronous data requests and retrieval to request units of information comprising less than an entire page. This has two very important benefits: much less information needs to be sent "through the wire," and the browser can continue working with other portions of a page while waiting for a response from the server.
- Heavy emphasis on client-side processing, to eliminate as many round trips as possible and to greatly improve the performance of the application.
- JavaScript binds everything together. AJAX takes advantage of the industry standard scripting language that is implemented by virtually every browser on every desktop.

They did so by combining the power, speed, and flexibility of AJAX with the drag-and-drop simplicity of ASP.NET. They created a library of AJAX controls that are as easy to use as the server-side ASP.NET controls we've been using since the Middle Ages. Even more important, they made it relatively easy to create drag-and-drop AJAX controls, although that's beyond the scope of this chapter.

This means you can get started using Microsoft's AJAX controls without first learning how to program in JavaScript or how to write DHTML. That lowers the usability bar enough that there really is no reason not to integrate AJAX into all your ASP.NET applications immediately.

Don't panic if you *like* JavaScript and you *want* to write your own AJAX controls; you are still free to do so. Just as with custom controls, you can always extend—or even invent—if you are so moved.

Now, you can eat your cake and have it, too. You can continue to create ASP.NET applications using the same IDE, but add client script with asynchronous postbacks (especially asynchronous data retrieval!), and you can do so with a library of tested, ready-to-use controls that fully encapsulate all the JavaScript for you.

The key point, however, is that asynchronous updates improve the performance of your application and the user's perception of that performance. This is because the page is not posted back to the server, but instead data is retrieved independently of the page being recreated, thus there is no flicker, and data retrieval is far faster.

ScriptManager and UpdatePanel

Microsoft realized that the job of integrating the standard ASP.NET controls and pages with AJAX controls (that encapsulate JavaScript and DHTML) would be difficult, tedious, and repetitious. So, they did it for you with the `ScriptManager` control, ensuring that you have access to a fully tested, reliable control that manages the grunt work. Adding a `ScriptManager` control to your page solves the problem, and having one on the page even when you don't need it comes at virtually no cost. Here is the declaration that must appear in every page:

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
```

The `ScriptManager` control will be visible in Design view, as shown in Figure 3-1, but will not be visible when the web site is run.

Implementing partial-page updates is surprisingly easy using ASP.NET AJAX—you just leave drag a `ScriptManager` control onto the page and leave its `EnablePartialRendering` property set to its default value of `True`.

Having done the hard work of *not* changing that property, you can then drag one or more `UpdatePanel` controls onto your page. Each `UpdatePanel` is updated individually and asynchronously, without affecting one another or anything else on the page.

That's it. Instant and unmistakable performance enhancement with almost no programmer effort.

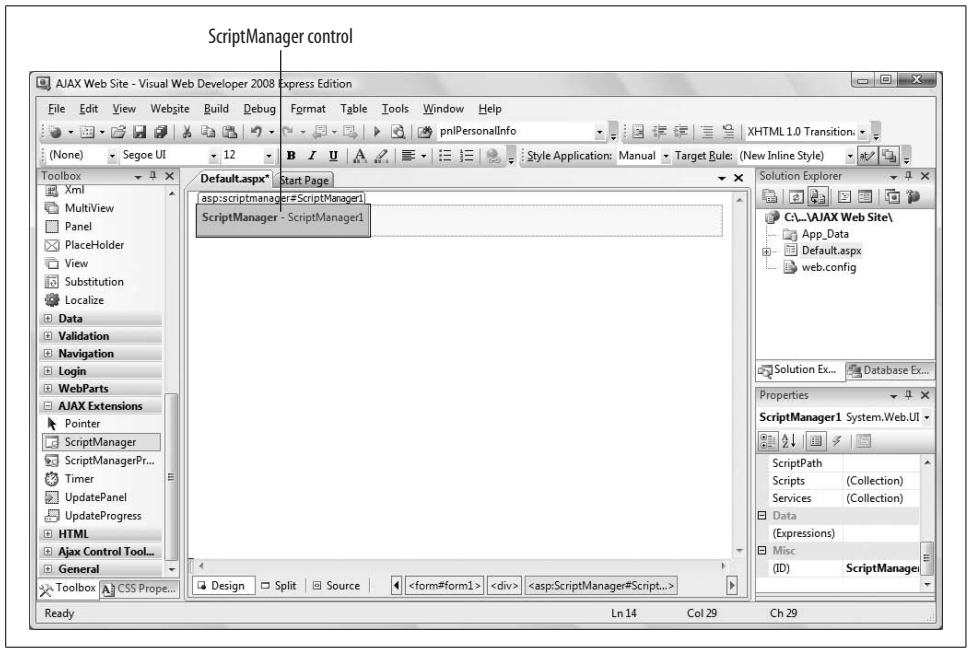


Figure 3-1. *The ScriptManager control is visible on the page in Design view, but you won't see it in the browser.*

To see this effect, you will modify the AdventureWorks project from the previous chapter, using update panels to improve performance. Recall in that example (shown in Figure 2-9), a pair of radio buttons was created (but never fully enabled) to control the visibility of a `Panel` control whose purpose was to collect personal information. You'll enable that feature now.

The way that you created the example in Chapter 2, clicking on the radio buttons causes a postback because the radio butttons' `AutoPostBack` property is set to `true`. This would cause the page to flicker as it was redrawn. With an AJAX `UpdatePanel`, however, the postback and update will be done asynchronously, and there will be no page flicker.

Begin by making a copy of the AdventureWorks Order Form site. Call it AdventureWorksRevisited. Run it to ensure that it works as expected.

See Appendix B for instructions on copying a web site.

To simplify the page, remove all the controls below the `pnlPersonalInfo` panel (everything below that `Panel` down to—but not including—the closing `div` and `form` tags).

You will also need to delete the no-longer relevant event handlers from the code-behind. (You can easily tell which code-behind methods are no longer relevant by trying to run the web page and looking at the build errors.) As it turns out, this includes all the event handlers from the AdventureWorks form.

Drag a `ScriptManager` control from the AJAX Extensions section of the Toolbox onto the page in either Source or Design view. It doesn't matter too much where it is, as long as it is inside the opening `<form>` tag, but the top of the page is a good choice.

In this example, you will add an AJAX `UpdatePanel` to surround the radio buttons and the `Panel` containing the personal information. The finished application is shown in Figure 3-2, which shows the panel both visible and hidden.

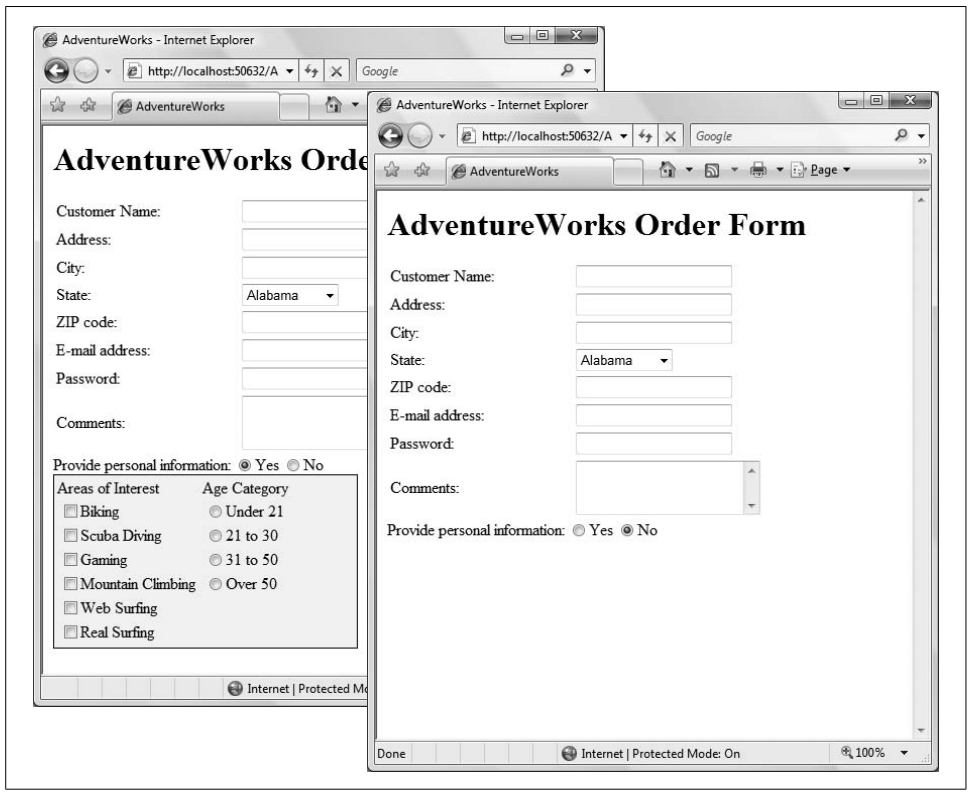


Figure 3-2. *The AdventureWorksRevisited web site with the Panel visible and hidden. You'll see that the AJAX version works much more smoothly than the version from Chapter 2.*

First, let's give the Yes and No RadioButtons something to do. Select each of the RadioButton controls and take a look at the Properties window. First, be sure the Text property is set correctly for each button—Yes or No. Next, make sure that the AutoPostBack property for each RadioButton is set to true. As we mentioned in Chapter 2, when AutoPostBack is set to true, the RadioButton immediately initiates a postback to the server when it is clicked, executing any action that should happen.

If you were to inadvertently double-click on one of the RadioButton controls, instead of single clicking to select it, then an event handler method would be created for the CheckChanged event. This is *not* what you want to happen at this point, because as you will see in a moment, you will want both radio buttons to call the same event handler. If it does happen, just go to the code-behind file and delete the just-created empty event handler method.

As you saw in Chapter 2, you indicate what action should be taken with event handlers. In Design view, set the event handler for the Yes button by single-clicking on the rbYes radio button. In the Properties window, click on the lightning bolt button to switch from properties to events. You will see that one of the events is CheckedChanged.

In the space to the right of the event name, enter the text YesNoEventHandler. Press tab (or Enter) to open the code-behind file, with the cursor positioned in the skeleton of the new event handler, ready for you to enter your custom code. Before you fill in the code for the event handler, though, return to Design view and single click on the rbNo radio button. Again, click in the space next to the same CheckedChanged event handler. This time, a down arrow will appear. Clicking that arrow will give you the opportunity to select an existing event handler, as shown in Figure 3-3.

Select YesNoEventHandler. Once again, the IDE will take you directly to the event handler in the code-behind file. Notice that the Handles statement now shows that this event handler handles the CheckChanged event for both radio buttons. Add the text shown highlighted in the following code snippet:

```
Protected Sub YesNoEventHandler( _
   ByVal sender As Object, _
   ByVal e As System.EventArgs) _
   Handles rbYes.CheckedChanged, rbNo.CheckedChanged
      pnlPersonalInfo.Visible = rbYes.Checked
End Sub
```

Run the program and click the Yes and No radio buttons in turn. You should see that the panel is displayed when the Yes button is checked, and hidden when the No button is checked. As you saw in the previous chapter, the Visible attribute specifies whether the panel is rendered, and now you've tied it to the value of the rbYes control.

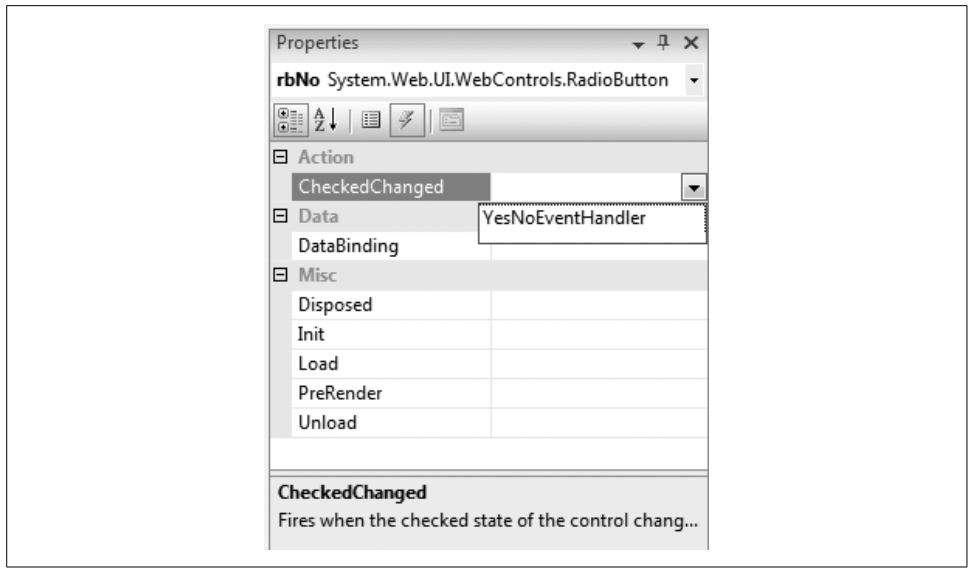


Figure 3-3. *Clicking the arrow next to the property of the CheckChanged control will let you wire the control to an event handler that's already created.*

As you click each button, however, you will probably notice a distinct flicker of the display as the entire page is redrawn. There may even be a detectable delay. This is because each time you click the button, the entire page is sent back to the server for processing, the event handler is run, and the entire page is sent back to the browser and redrawn.

AJAX solves this problem by asynchronously updating only portions of a page. By updating segments "in the background," you avoid reposting and redrawing the entire page.

In the AdventureWorks example as it is currently written, this postback also causes the user's position on the page to be lost. When the page is rendered from scratch, the browser effectively scrolls back to the top of the page, which can be very annoying.

You can rectify this by setting the `MaintainScrollPositionOnPostback` property of the `Page` directive to `true`. To do so, open the markup file in Source view and edit the `Page` directive at the top of the file, adding the following highlighted code:

```
<%@ Page Language="VB" AutoEventWireup="true"
   CodeFile="OrderForm.aspx.vb" Inherits="_Default"
   MaintainScrollPositionOnPostback="true"%>
```

Select the `ScriptManager` control in Design view and look at the Properties window. If the Properties window is not visible, right-click on the control and select Properties or press F4. Verify that the `EnablePartialRendering` property is set to `True`.

Your goal is to update only the Panel named pnlPersonalInfo when the user clicks one of the radio buttons. To do that, you need to *wrap* the Panel and the radio buttons inside an UpdatePanel control, which you'll find in the AJAX Extensions section in your Toolbox.

Make sure that you are in Design view, open the AJAX Extensions tab of the Toolbox, and drag an UpdatePanel onto the form (you can do the same in Source view, of course). Now, highlight the text prompt, the radio buttons, and the pnlPersonalInfoPanel, and drag them all onto that UpdatePanel. That's all there is to it.

If you look at the resulting page in Source view, you will see not only the UpdatePanel elements, but also opening and closing <ContentTemplate> tags inside the UpdatePanel. All of your content must be inside these <ContentTemplate> tags. For example, if you had a Button and a TextBox inside an UpdatePanel, the markup would look like this:

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
   <ContentTemplate>
      <asp:Button ID="Button1" runat="server" Text="Button" />
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
   </ContentTemplate>
</asp:UpdatePanel>
```

Doing this in Source view is very similar: drag the UpdatePanel from the Toolbox onto the window, and then move the relevant markup inside the UpdatePanel. However, if you do this in Source view, the <ContentTemplate> tags will not be created automatically, and you must manually type them into the markup.

Run the program again, and then click the radio buttons to see the difference. This time, there should be no flicker as only the panel reloads. Feel free to say "Wow!"

Controlling Browser History

It has been said that "he who controls history, controls the future." That is certainly true when using a browser. Everyone is familiar with the browser Forward and Back buttons, which allow the user to navigate back and forth amongst the pages already visited in the current session. Forward and Back are central to the browsing experience.

The browser automatically enables and disables these buttons as necessary, if there are appropriate pages to navigate to. For example, if you are at the first page visited that session, then the Back button will be disabled because you can't go back any further.

The browser maintains a history which contains a list of all the URLs sent back to the server as page requests. However, it is here that the benefits of AJAX turn around

and bite us. With asynchronous postbacks, the URL is not posted back to the server as a page request. This means that AJAX asynchronous postbacks are not added to the browser's history, do not affect the Forward or Back buttons, and are not accessible to those buttons or in any history lists. Bummer.

You can see this in the previous example, AdventureWorksRevisited. Run that example again, and notice that the Forward and Back buttons in your browser are disabled—that's because this is the only page you've visited so far. Now change the radio buttons that control the display of personal information. Even though the content of the page appeared to change, neither the Forward nor Back buttons are enabled. This is because a full page request is not being made of the server, so the browser doesn't know that there's been a request and response from the server.

Out in the real world of deployed web sites, if the navigation buttons are enabled as a result of previously navigating to other pages, they will not take you where you intuitively think you ought to go if you've used any AJAX-enabled controls. And worst of all in that case, you will lose any work you have done in any AJAX-enabled controls on the page.

With ASP.NET 3.5 Service Pack 1, you can fix this, although it does take some coding on your part (something we try to avoid as much as possible in this book). There are two things you need to deal with. The first is to create a *history point* and add it to the browser history when the state of the page changes. The second is to handle the clicks when the user clicks the brower's Forward and Back buttons or history lists and to restore the state of the page as it was earlier. The `ScriptManager` control included on every ASP.NET AJAX web page, along with the `UpdatePanel` control, provides the means to accomplish both of these ends.

An example will show how this works. Create a new ASP.NET web site called AjaxHistory.

Before you can manage browser history, you must add and configure the `ScriptManager` control. In either Source or Design view, drag a `ScriptManager` control from the AJAX Extensions section of the Toolbox onto the page. Then, put the cursor on the `ScriptManager` control so that its properties are visible in the Properties window. Set the `EnableHistory` property to `True`, as shown in Figure 3-4. With this property set, history is enabled for all AJAX-enabled controls on the page, that is, for all controls inside `UpdatePanel` controls. The default setting of this property is `False`, which is why the Forward and Back buttons were disabled when you tried them a moment ago.

Drag an `UpdatePanel` control from the AJAX Extensions section of the Toolbox onto the page. You will place all the other controls whose history is to be tracked inside this `UpdatePanel`. For this example, the page will contain a `RadioButtonList` with

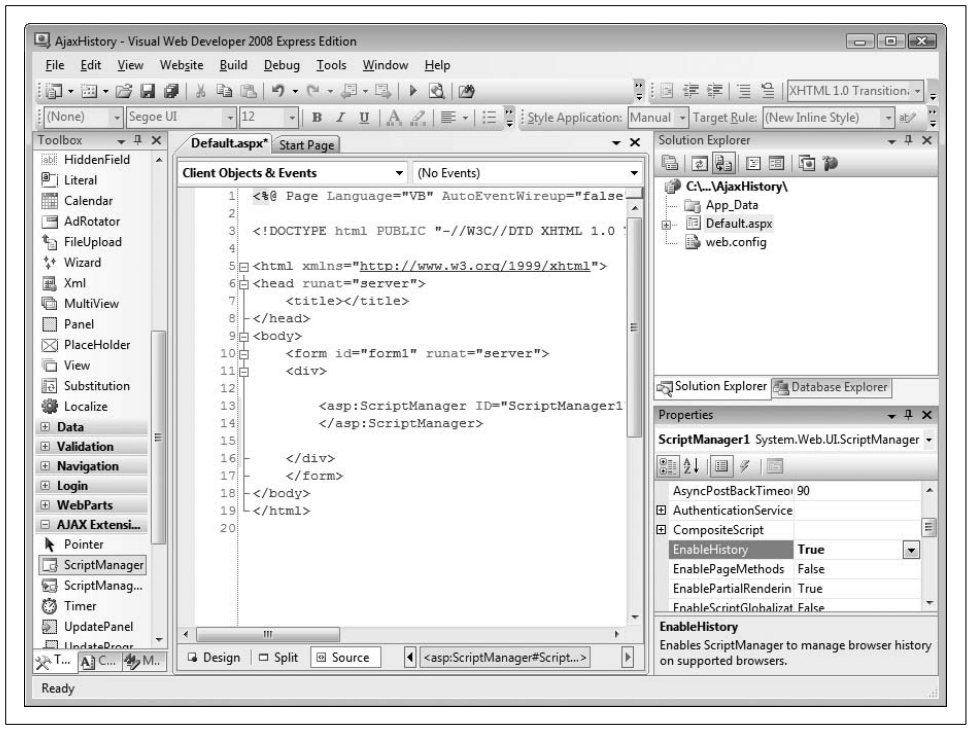


Figure 3-4. *Set the EnableHistory property on the ScriptManager control to turn on history control for all the controls inside UpdatePanel controls anywhere on the page.*

three options, a `ListBox` with three items, plus a button for updating the history. The finished page is shown in Figure 3-5.

Switch to Design view. Drag a `RadioButtonList` from the Standard section of the Toolbox inside the `UpdatePanel`. Click on the Smart Tag of the `RadioButtonList` and select Edit Items. Add three items, as shown in Figure 3-5. Let the IDE set the default `Value` property of each `ListItem` to be the same as the `Text` property. With the `RadioButtonList` still selected, set the `AutoPostBack` property to `True`, so the control will force an immediate postback whenever a change is made. Remember, because this control is inside the `UpdatePanel`, that postback will be an asynchronous, partial page postback.

Drag a `Label` control onto the page. This will display the value of the selected radio button every time a different radio button is selected. To make this happen, double-click on the `RadioButtonList` control in Design view to open an event handler method in the code-behind. Add the highlighted line of code in Example 3-1.

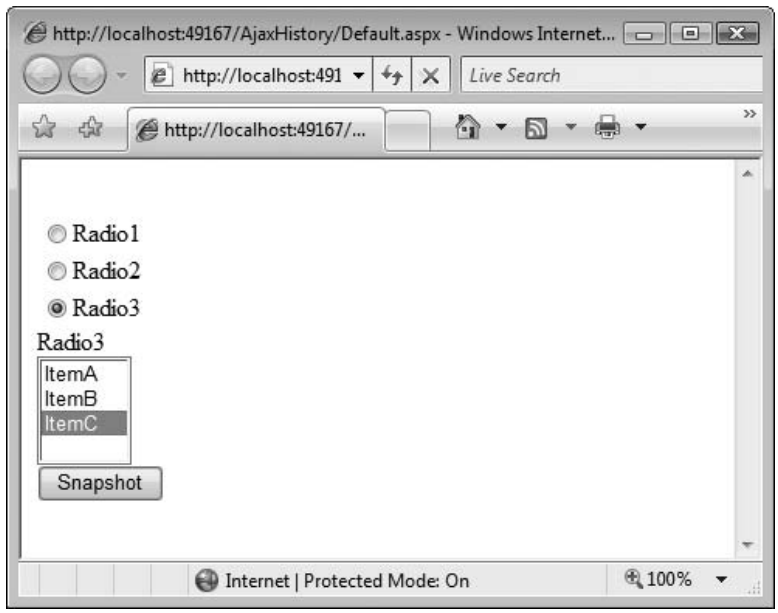


Figure 3-5. *The sample page for exercising the browser history with AJAX controls looks like this.*

Example 3-1. *The SelectedIndexChanged event handler of the RadioButtonList updates the label with the new selected value*

```
Protected Sub RadioButtonList1_SelectedIndexChanged(ByVal sender As Object, _
                                                    ByVal e As System.EventArgs) _
                                                    Handles RadioButtonList1.
SelectedIndexChanged
    Label1.Text = RadioButtonList1.SelectedValue
End Sub
```

Next, switch back to *Default.aspx* in Design view, and drag a `ListBox` control from the Standard section of the Toolbox onto the page, inside the `UpdatePanel`. You might first want to hit Enter after the `Label` to make some space. Click the Smart Tag of the `ListBox` and click on Edit Items, just as you did for the `RadioButtonList`. This time, add the three items shown in the `ListBox` in Figure 3-5. Because we are not doing anything with the selected value anywhere in this example, there is no need to set `AutoPostBack` to `True`, nor is there any code-behind associated with this control.

There are two general strategies you can follow. The first would be to create a history point whenever a control changes state. In this case, you would add code to the default event handler for any (or every) control of interest. This works well when you are only tracking a single control on the page, but gets cumbersome if you want to track multiple controls, as is the case with this example.

The second strategy, which we'll demonstrate in this example, only creates a history point when the user clicks a button, at which point the state of all the relevant controls is saved. To implement this, drag a `Button` control onto the page, again inside the `UpdatePanel`, underneath the `ListBox`. Change its `Text` property to "Snapshot". Double-click on the `Button` to get the event handler for the `Click` event. Add the highlighted code from Example 3-2.

Example 3-2. *Clicking on the Button creates a history point for each of the controls on the page*

```
Protected Sub Button1_Click(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
                            Handles Button1.Click
    Dim strRadioButtonListValue As String = RadioButtonList1.SelectedValue
    Dim strListBoxValue As String = ListBox1.SelectedValue
    Dim strTitle As String = "History: " + strRadioButtonListValue + _
                                    " - " + strListBoxValue

    ScriptManager1.AddHistoryPoint("RadioButtonList", _
                                   strRadioButtonListValue, strTitle)
    ScriptManager1.AddHistoryPoint("ListBox", _
                                   strListBoxValue, strTitle)
End Sub
```

This code, which runs every time the button is clicked, gets the selected values of the two controls, assigning them to `String` variables. It also constructs a `String` variable, `strTitle`, from a concatenation of the two selected values.

Next, you call the `AddHistoryPoint` method of the `ScriptManager` control twice, once for each of the controls you are interested in preserving, and passing in the `strTitle` string that you created. The `AddHistoryPoint` method, as its name implies, adds a history point to the `History` object. There are several variants, or *overloads*, of this method. The one used here takes three arguments:

```
ScriptManager1.AddHistoryPoint("RadioButtonList", strRadioButtonListValue, strTitle)
```

The first argument is the *key* that identifies this history point. A key is simply an identifier that can be associated with a value. Each history point that you save will have its own key that you can use later to retrieve that history point state. In this line of code, the key is hardcoded to be `RadioButtonList`. Later, by retrieving the state named `RadioButtonList`, you'll be able to set the radio buttons to their earlier state.

The second argument is the value associated with this key. This second argument is the state that you want to save at that moment. This typically would be information such as which radio button is selected, what item is selected in a list, what the value is in a text box, and so on. It's up to you to determine what state you want to save, and also how you want to save it. There is no predefined format for state data; you will retrieve it later during navigation, so you can use any format that will help you restore the state later if needed.

In this example, the state to be saved for both controls is the value of the `SelectedValue` property. This way, the retrieval method (which you'll see in a moment) can use the key to refer to the selected item. The third argument is the title, which will be displayed when the browser displays history lists. The title is a string concatenation of a literal with the variables naming the selected items:

```
Dim strTitle As String = "History: " + strRadioButtonListValue + _
                         " - " + strListBoxValue
```

With this code in place, history points will be added to the browser's history whenever the user clicks the Snapshot button.

That's a start, but this code doesn't tell the Forward and Back buttons what to do automatically. For that, you need to handle the `Navigate` event. You saw the default `Click` event for a `Button` control above, and the `Navigate` event isn't much different. When the user clicks on the Forward or Back button or selects an entry from one of the history lists provided by the browser, the `ScriptManager` raises the `Navigate` event.

To handle this event, switch back to *Default.aspx* in Design view, and select the `ScriptManager` control. In the Properties window, click the lightning bolt button to see all the available events. Double-click in the box next to the `Navigate` event.

You will be taken to the code editor, inside a newly constructed event handler, ready to type your code. Add the highlighted code from Example 3-3.

Example 3-3. *Handle the Navigate event to process the Forward and Back buttons and history lists*

```
Protected Sub ScriptManager1_Navigate(ByVal sender As Object, _
                                      ByVal e As System.Web.UI.HistoryEventArgs) _
                                      Handles ScriptManager1.Navigate
    ' set the value of the RadioButtonList
    Dim strRadioButtonList As String = String.Empty
    If Not e.State("RadioButtonList") Is Nothing Then
        strRadioButtonList = e.State("RadioButtonList")
    End If

    If (strRadioButtonList.Length > 0) Then
        RadioButtonList1.SelectedValue = strRadioButtonList
    Else
        RadioButtonList1.SelectedIndex = -1
    End If
    Label1.Text = strRadioButtonList

    ' set the value of the ListBox
    Dim strListBox As String = String.Empty
    If Not e.State("ListBox") Is Nothing Then
        strListBox = e.State("ListBox")
    End If

    If (strListBox.Length > 0) Then
        ListBox1.SelectedValue = strListBox
    Else
```

Example 3-3. *Handle the Navigate event to process the Forward and Back buttons and history lists (continued)*

```
        ListBox1.SelectedIndex = -1
    End If

    Page.Title = "History - " + strRadioButtonList + " - " + strListBox
End Sub
```

Notice the second argument in the method declaration is of type `HistoryEventArgs`. This parameter carries information about the history point that is being navigated to. `HistoryEventArgs` has a single property called `State`. `State` is a `NameValueCollection` object, which means that it consists of one or more pairs of strings. The first half of each pair is the *key*, or *name*, of the pair. The second half of each pair is the *value* of the pair. Together they constitute a *name/value pair*. If you know the name, you can retrieve the value by using the name as an *index* into the `State` collection. That is, you retrieve the value of the pair in the `State` collection for the `RadioButtonList` with the name `"RadioButtonList"`, and assign it to a new string variable, `strRadioButtonlist`. You then do the same for the `ListBox`. In either case, you always want to test if that `State` collection exists before trying to read it, otherwise an error will result. The following snippet achieves this. The string variable is declared outside the `If-Then` block, so it will be available to the code further down whether or not the `State` collection exists:

```
Dim strRadioButtonList As String = String.Empty
If Not e.State("RadioButtonList") Is Nothing Then
    strRadioButtonList = e.State("RadioButtonList")
End If
```

Back in Exercise 3-2, when you called `AddHistoryPoint`, the first parameter passed in was the key, and the second parameter was the value of the `State` property. Now you've retrieved the values of those history points, and assigned them to `strRadioButtonList` and `strListBox`. With this information, you can set the selected value of the `RadioButtonList` and it's associated `Label.Text` property, as well as the selected value of the `ListBox`. If nothing was selected for either control, set the index to `-1`, which has the effect of selecting nothing.

The last thing this event handler does is set the Title of the page.

The end result is shown in Figure 3-6 after clicking on the radio buttons several times and showing the history list under the Forward button.

Notice that the title of the browser window reflects the title information that you passed when you called `AddHistoryPoint`. More interestingly, notice that the URL in the Address box contains not just the name of the page (*Default.aspx*), but a string of characters that look like gibberish. That's your saved state. It's encoded in a way that allows it to be added to a URL. As you navigate back and forth in the browser, you'll see that this state string changes (although the base page name does not). And each time you navigate, that state is sent to the server, where you handle it in the `Navigate`

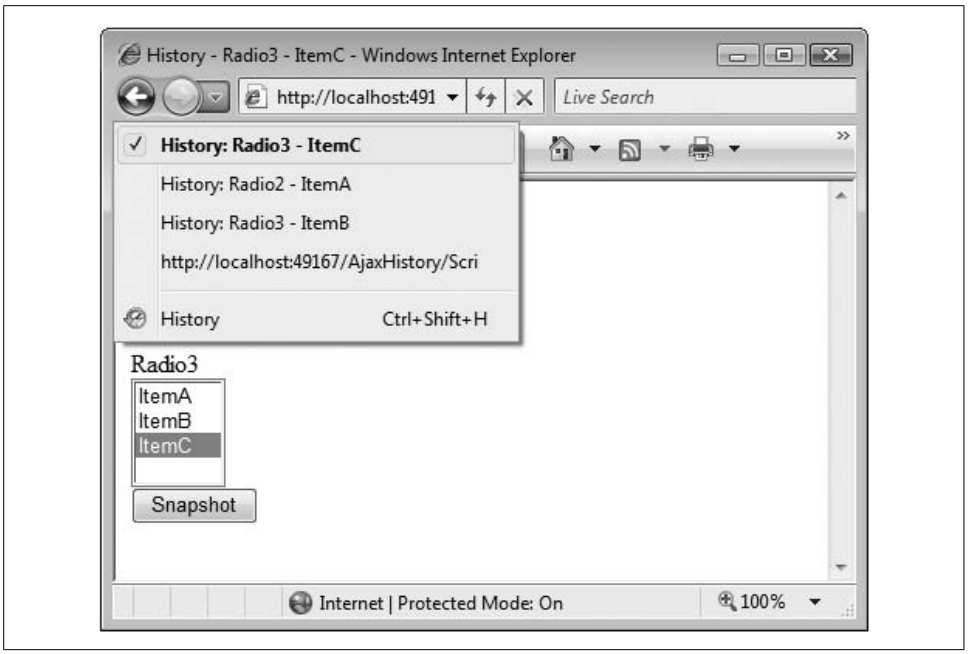


Figure 3-6. *The Back and Forward buttons work as expected, along with history lists.*

event. This really is the whole secret of how ASP.NET manages AJAX navigation. ASP.NET creates a new URL that includes your state each time your page performs an asynchronous postback, and it adds the new URL to the browser's history automatically. When you click the browser's Back button, the browser simply goes back one item in its history, just like it always does. Now, though, the browser's history contains URLs with all the history points that have been saved, and the browser ends up always doing the right thing during navigation.

It might be obvious, but being able to restore state during navigation does not allow you to magically reverse the effect of something like a database update. (Unless in your code for the Navigation handler you implement this functionality yourself.) As you've seen, browser history isn't really doing anything as such; it's simply storing information for you that you might need later in response to user navigation. This is a very welcome feature in ASP.NET AJAX, but it does require that you think carefully about what constitutes state at any given point in your web application, and about what information you need to save.

Extending Controls with the Control Toolkit

The AJAX Control Toolkit is a shared-source collection of pre-built AJAX controls, available for download from the Microsoft web site, which integrates directly into the IDE. Before you proceed with this section, make sure you have installed the

AJAX Control Toolkit, as described in Appendix A. These controls are developed by the community, so the list of available controls changes regularly, and the controls may not have the same reliability you'd expect from the built-in ASP.NET controls. They're also not officially supported by Microsoft, but you can get great support from the community.

The AJAX Control Toolkit provides a number of additional AJAX-enabled controls you can use to enhance the functionality of your web application. Some of the more useful controls in the Control Toolkit are listed in Table 3-1.

Table 3-1. *A sample of the AJAX Toolkit controls*

Toolkit control	Description
`Accordion`	A control that provides multiple panes, only one of which is visible at a time
`AlwaysVisibleControlExtender`	Keeps a control visible as the user scrolls the page
`Calendar`	Can be attached to an ASP.NET `TextBox` to provide a pop-up calendar control for selecting a date to enter in the `TextBox`
`CascadingDropDown`	The user's selection from one drop down control determines the choices available in the next drop down
`CollapsiblePanelExtender`	Allows any Panel to collapse and expand
`ConfirmButtonExtender`	When the user clicks a button, a dialog box pops up to confirm the choice
`DragPanelExtender`	Lets the user drag a panel around on the page
`DropShadow`	Applies a drop shadow effect to an ASP.NET `Panel` control
`FilteredTextBoxExtender`	Ensures that only "valid" text may be entered into a `TextBox`
`HoverMenuExtender`	Pops up a menu when the mouse hovers over a control
`MutuallyExclusive-CheckBoxExtender`	Pick none or one of several checkboxes; this provides functionality similar to radio buttons, but with the ability to uncheck all the checkboxes
`NoBot`	A control which attempts to prevent spam or robot interaction with a web site
`PasswordStrength`	Helps the user pick a good password
`PopupControlExtender`	Can be attached to any control to provide a pop-up window with additional content relevant to the attached control
`ReorderList`	Lets the user reorder the members of a list by dragging them into place
`RoundedCornersExtender`	Applies rounded corners to a target control
`Slider`	Extends a `TextBox` control with a horizontal or vertical slider for selecting a numeric value from a range of values
`TextBoxWaterMarkExtender`	Displays helpful text in a textbox until you start to type

TextBoxWaterMarkExtender

Many of the Toolkit controls are "extenders"—that is, rather than acting alone, they extend the behavior of one of the standard controls. For example, the `TextBoxWaterMarkExtender` works with a `TextBox` to add the watermark effect. The extender has properties to identify which `TextBox` it will modify, what text will be used as the watermark, and what style should be applied to the watermark text itself. Figures 3-7 and 3-8 demonstrate watermarks in action.

Figure 3-7. *This is what the watermarked control looks like before the user enters any data. The watermark serves as a reminder of what the user should enter, and makes it clear that the field is currently empty.*

Figure 3-8. *When the user types in the TextBox, the watermarked style is applied, which shows an obvious change from the watermarked style.*

To demonstrate the watermark effect, just copy the previous example, called AdventureWorksRevisited, to a new web site called AdventureWorksWaterMarks.

Before modifying the page, you need to create a style sheet that will specify the styles for the watermarked and unwatermarked text.

Styles and style sheets are explained fully in Chapter 6, so we will only show the bare basics here.

To create a style sheet, click Website → Add New Item…. In the Add New Item dialog box, select Style Sheet, accept the default name of *StyleSheet.css*, and then click the Add button, as shown in Figure 3-9.

This will open a style sheet in the editor with an empty `body` element. Add the highlighted code from Example 3-4 to this style sheet.

In this style sheet, you're adding two style classes, `watermarked` and `unwatermarked`, that will be applied to the text in the example. You don't need a specific style sheet to use the `TextBoxWaterMarkExtender`, but it makes things easier.

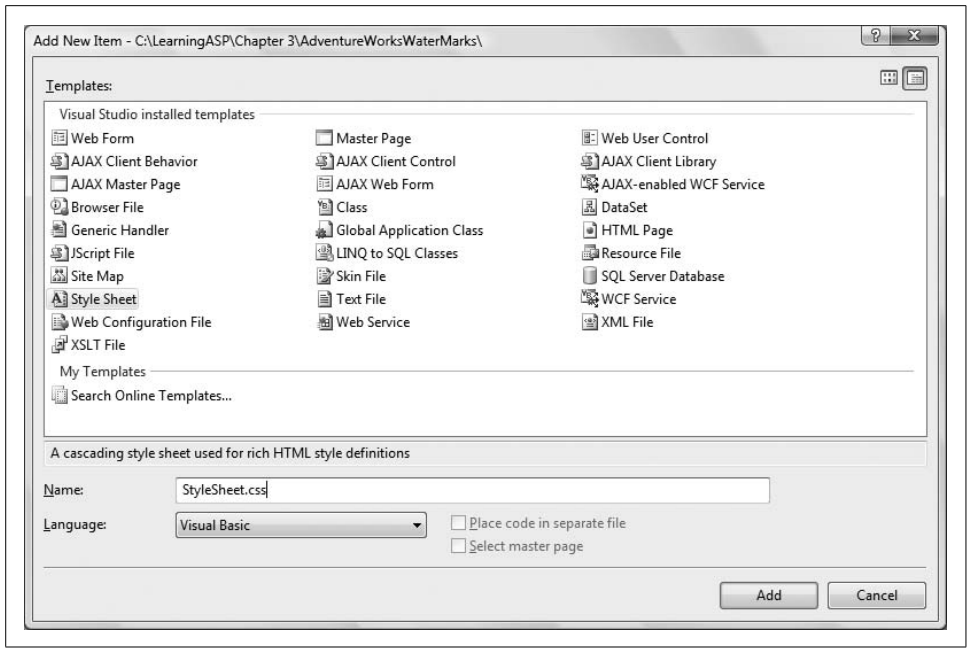


Figure 3-9. *To add a style sheet to the web site, use the Add New Item dialog box.*

Example 3-4. *This StyleSheet file is added to your project to define the styles for the watermarks; you don't need to know what the exact styles do right now*

```
body {
}
.watermarked {
    padding:2px 0 0 2px;
    border:1px solid #BEBEBE;
    background-color:#F0F8FF;
    color:Gray;
    font-family:Verdana;
    font-weight:lighter;
}
.unwatermarked {
    height:18px;
    width:148px;
    font-weight:bold;
}
```

Next, in Design view, go to *OrderForm.aspx*. Select the Customer Name Text Box, `txtName`, and in the Properties window, set the `CssClass` property to `unwatermarked`. (The style class names are case sensitive.) Do the same for `txtAddress`. This sets the style class that will apply to the text the user types into these text boxes, as illustrated previously in Figure 3-8.

Click on the Customer Name `TextBox` in Design view to bring up the smart tag, and then click on the tag and select Add Extender… from the drop-down menu.

You will see an Extender Wizard dialog box. Slide along until you find the TextBoxWatermarkExtender, and select it. Before clicking OK, notice the default ID assigned to it, as shown in Figure 3-10. You could change it, but it is a reasonable name, so we will leave it as is.

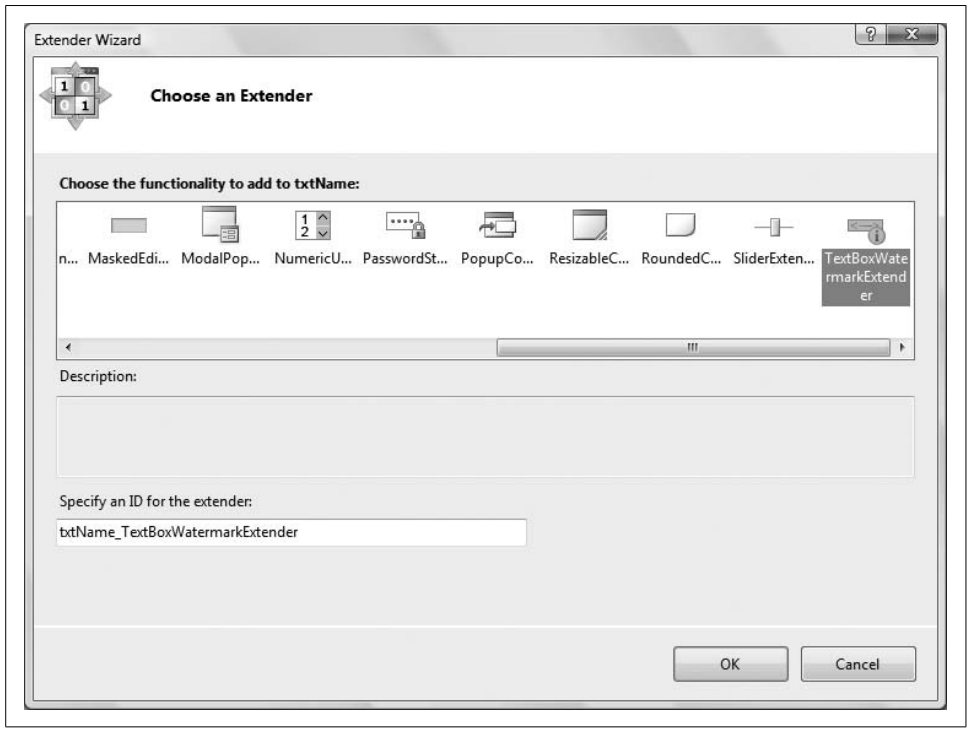


Figure 3-10. *The Extender Wizard provides a convenient list of the available extenders you can use. When you select one, the wizard generates a default ID for you.*

If you switch to Source or Split view, you can see that the wizard automatically sets the `ID` property of the control to `txtName_TextBoxWatermarkExtender` and the `TargetControlID` property to `txtName`. The `ID` property is the same as the `ID` properties for all the other controls you've seen so far, but the `TargetControlID` property (which, strangely enough, is not visible in the Properties window for the control) specifies the control that you want the watermark effect to apply to—in this case, it's the `TextBox` control, `txtName`. Add a similar `TextBoxWatermarkExtender` to the Address `TextBox`; the screen should look something like Figure 3-11.

There are two other properties you need to set for this control: `WatermarkCssClass` and `WatermarkText`. Unfortunately, these properties are not accessible through the Properties window, so you need to switch to Source view to manually type them in.

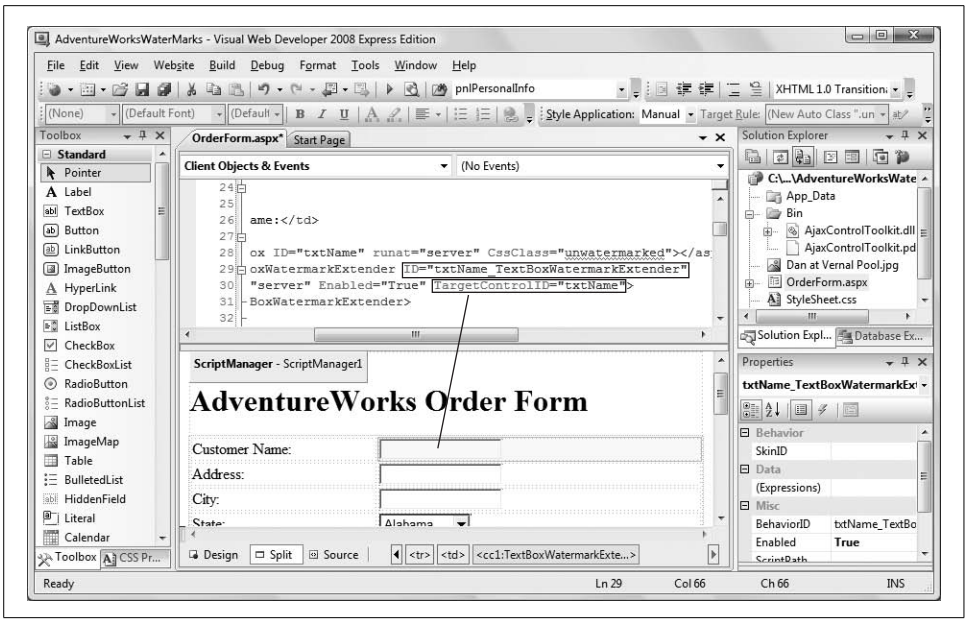


Figure 3-11. *When you add the TextBoxWatermarkExtender control to the form by using the Smart Tag on the TextBox, the wizard automatically sets the TargetControlID property, and suggests an ID property.*

Switch to Source view and add the `WatermarkCssClass` and `WatermarkText` attributes. Add those two lines of code to each control, so that the result looks like Example 3-5.

Example 3-5. *Add two properties to the TextBoxWatermarkExtender in the markup file*

```
<tr>
    <td style="width: 100px">
        Customer Name:
    </td>
    <td style="width: 150px">
       <asp:TextBox ID="txtName" runat="server"
           CssClass="unwatermarked" >
       </asp:TextBox>
       <cc1:TextBoxWatermarkExtender
          ID="txtAddress_TextBoxWatermarkExtender"
          runat="server"
          Enabled="True"
          TargetControlID="txtName"
          WatermarkCssClass="watermarked"
          WatermarkText="Your name" >
       </cc1:TextBoxWatermarkExtender
     </td>
</tr>
<tr>
```

Example 3-5. *Add two properties to the TextBoxWatermarkExtender in the markup file (continued)*

```
    <td style="width: 100px">
        Address:</td>
    <td style="width: 150px">
       <asp:TextBox ID="txtAddress" runat="server"
            CssClass="unwatermarked" >
       </asp:TextBox>
       <cc1:TextBoxWatermarkExtender
           ID="txtAddress_TextBoxWatermarkExtender"
           runat="server"
           TargetControlID="txtAddress"
           WatermarkCssClass="watermarked"
           WatermarkText="Home address" >
       </cc1:TextBoxWatermarkExtender
    </td>
</tr>
```

These two watermark attributes add the watermark itself to the `TextBox` control before the user types anything in. The `WatermarkText` property sets the text that will appear in the `TextBox`, and the `WatermarkCssClass` property applies the style class that you defined earlier in the style sheet. The result is that the `TextBox` fields have a nicely styled reminder text in them before the user types anything, as you saw in Figure 3-7.

One final step is to add the following line of HTML to the markup file, inside the `<head>` element to make the style sheet visible to the page. Without this line, none of the style classes you created earlier will apply to the page:

```
<style type="text/css">@import url(StyleSheet.css);</style>
```

Now run the page. The Customer Name and Address fields will look like Figure 3-7. When you type in the text box, the change is noticeable and removes any potential confusion, as shown previously in Figure 3-8.

PopupControlExtender

Screen real estate on a web page is often at a premium, so the `PopupControlExtender` is a very useful tool for presenting the maximum information in a minimum of space. You attach a `PopupControlExtender` to a control. When the user clicks that control, a pop-up window appears with additional content. If you put an `UpdatePanel` into the pop up, it can display data retrieved asynchronously from the server—a very powerful effect.

To see how you can make this feature work for you, you'll modify the previous example, enhancing the `RadioButtonList` used for selecting an age category with a `TextBox`. You'll add a `PopupControlExtender` and attach it to the `TextBox`. The `PopupControlExtender` will use an `UpdatePanel` to present the `RadioButtonList` as a pop up.

Figure 3-12 shows the TextBox waiting to be clicked on. To prompt the user to click inside the TextBox, there's also a TextBoxWatermarkExtender.

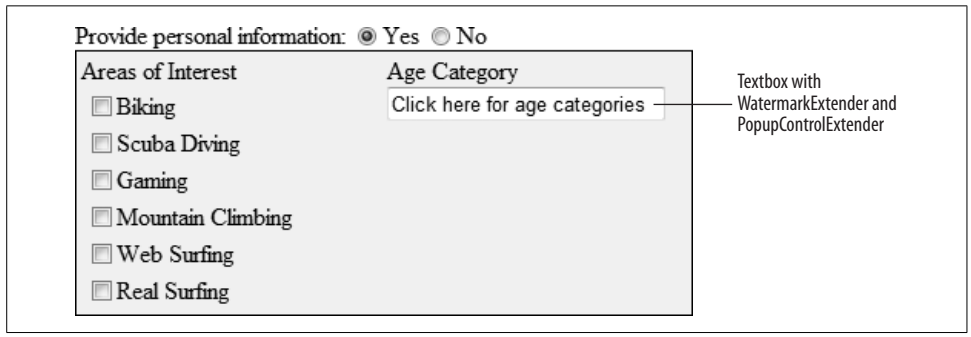


Figure 3-12. *The TextBox has a PopupControlExtender attached to it, and also a TextBoxWatermarkExtender to invite users to click it.*

Figure 3-13 shows what happens when the user clicks in the TextBox. The watermark disappears, and the pop-up window appears, containing the RadioButtonList showing the categories the user may pick from. Because the list is inside an UpdatePanel, there is no postback to the server (and thus no screen flicker); everything happens on the client side.

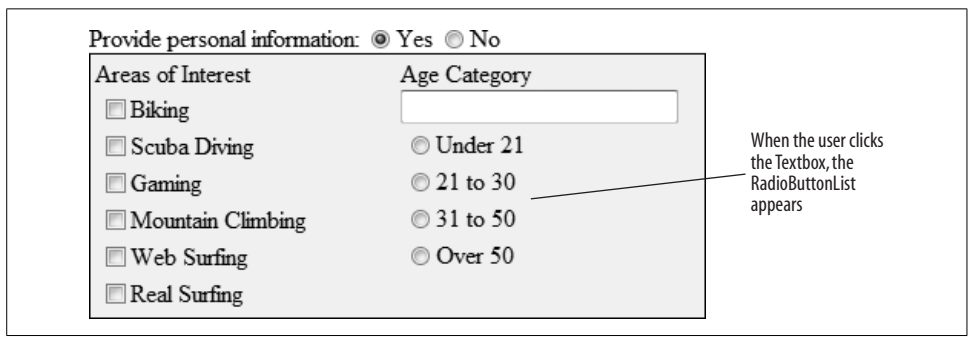


Figure 3-13. *When the user clicks on the TextBox with the PopupControlExtender, the pop-up panel opens, showing the radio buttons.*

When the user chooses a radio button, the choice is "posted back," but again, using an UpdatePanel, so the rest of the page is unaffected. The panel closes and the choice is displayed in the TextBox, as shown in Figure 3-14.

To implement this example, copy the previous example, AdventureWorksWatermarks, to a new web site, called AdventureWorksPopupControl. First, you'll do all the dragging and dropping and coding, and then we'll follow with an explanation of how it all works.

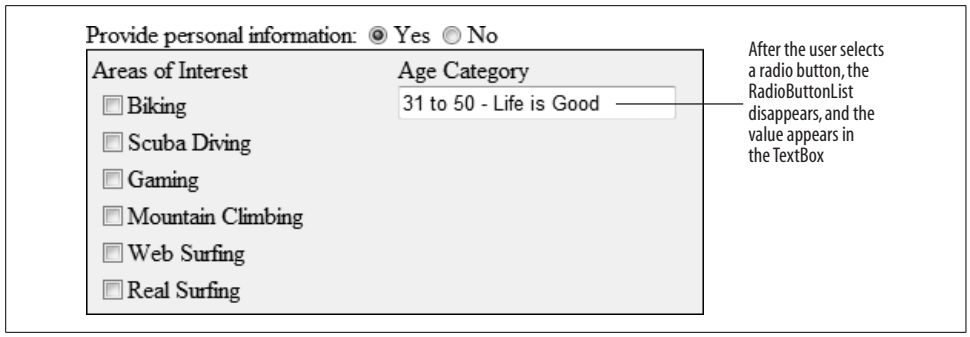

Provide personal information: Yes No
Areas of Interest
Biking
Scuba Diving
Gaming
Mountain Climbing
Web Surfing
Real Surfing
Age Category
31 to 50 - Life is Good
After the user selects a radio button, the RadioButtonList disappears, and the value appears in the TextBox

Figure 3-14. *After the user makes a selection, the UpdatePanel is closed and the TextBox is updated.*

The previous example had the following markup for the layout table cell containing the Age Category caption and radio buttons:

```
<td style="width: 1024px">
    Age Category
    <br />
    <asp:RadioButtonList ID="rblAge" runat="server"
                         AutoPostBack="True" Width="125px">
        <asp:ListItem>Under 21</asp:ListItem>
        <asp:ListItem>21 to 30</asp:ListItem>
        <asp:ListItem>31 to 50</asp:ListItem>
        <asp:ListItem>Over 50</asp:ListItem>
    </asp:RadioButtonList>
</td>
```

In Design view, drag a `TextBox` control from the Toolbox to the cell that currently contains `rblAge`. Set its `ID` property to `txtAgeCategory`, and its `Width` property to `175px`. Next, click on the Smart Tag of the `TextBox` and click on Add Extender. From the Extender Wizard dialog box, add a `TextBoxWatermarkExtender` to the `TextBox`, then click the Smart Tag again, and add a `PopupControlExtender`. You can accept the default IDs for these two extenders. Next, drag a standard `Panel` control into the cell and set its `ID` property to `pnlAgeCategories`.

Switch to Source view, and you'll see that the `TargetControlID` property of the `TextBoxWatermarkExtender` has already been set to `txtAgeCategory` (the ID of new `TextBox`). You also want to set its `WatermarkText` property to "Click here for age categories" (you need to set this property in Source view because extender controls are not selectable in Design view). You could also set the `WatermarkCssClass` property, as we did in the previous section, but we will not bother to do so here.

While still in Source view (again, not all the properties are visible in Design view), look at the properties of the `PopupControlExtender`. The `TargetControlID` is already set to the `ID` of the `TextBox` (`txtAgeCategory`). This will cause the pop up to appear when `txtAgeCategory` is clicked. Set `PopupControlID` to `pnlAgeCategories`. This is the control that will pop up when the `TextBox` is clicked (you will populate that `Panel` in

a moment). Finally, set the `Position` property of the `PopupControlExtender` to `Bottom`. (IntelliSense is a big help in setting these properties.)

You will also see the following attributes in the markup:

```
ExtenderControlID=""
DynamicServicePath=""
```

These are spurious and may either be left in place or deleted. We will delete them from our example for clarity.

Now you need to populate the `Panel`. This is easiest to do while remaining in Source view. From the AJAX Extensions section of the Toolbox, drag an `UpdatePanel` control into `pnlAgeCategories`. Manually enter an opening and closing `<ContentTemplate>` tag inside the `UpdatePanel`. (IntelliSense will help you.) Next, drag the pre-existing `RadioButtonList` (`rblAge`) inside the `UpdatePanel` you just placed within the `<ContentTemplate>` tags.

You could stop right here and this would work as is, but edit the `RadioButtonList` to add explicit `Value` properties to each of the items. In Design view, click on the Smart Tag of the `RadioButtonList`, then Edit Items to bring up the ListItem Collection Editor, as shown in Figure 3-15.

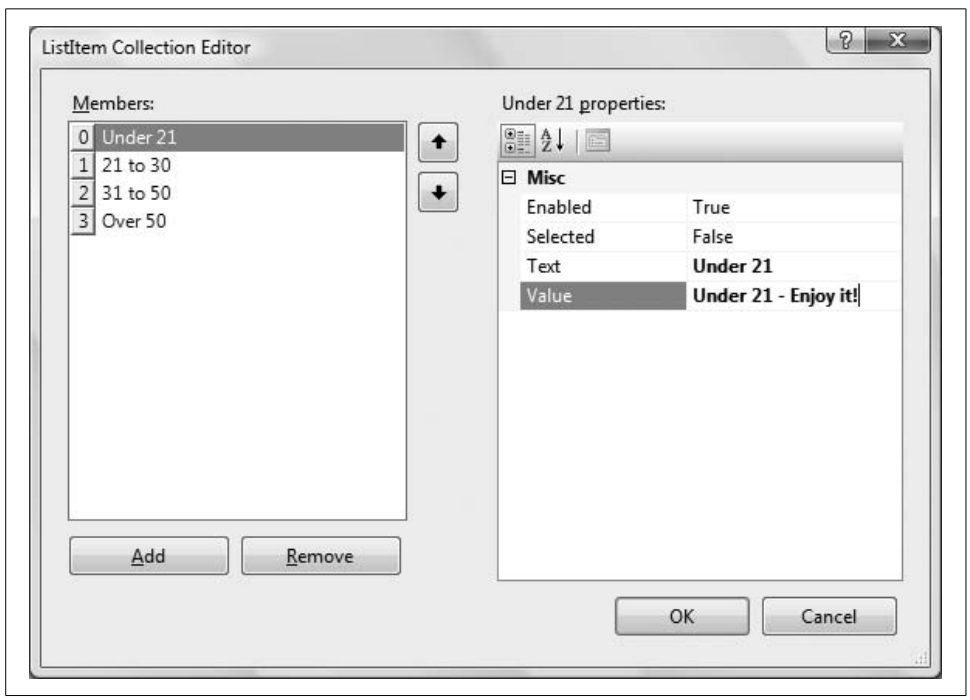


Figure 3-15. *After clicking the Smart Tag of the RadioButtonList, you get this ListItem Collection Editor for editing the items in the list.*

Click on each of the Members in turn, and change the Value properties as follows.

Text	Value
Under 21	Under 21 - Enjoy it!
21 to 30	21 to 30 - Livin' Large
31 to 50	31 to 50 - Life Is Good
Over 50	Over 50 - Golden Years

When you have modified all the values, Design view should look something like that shown in Figure 3-16.

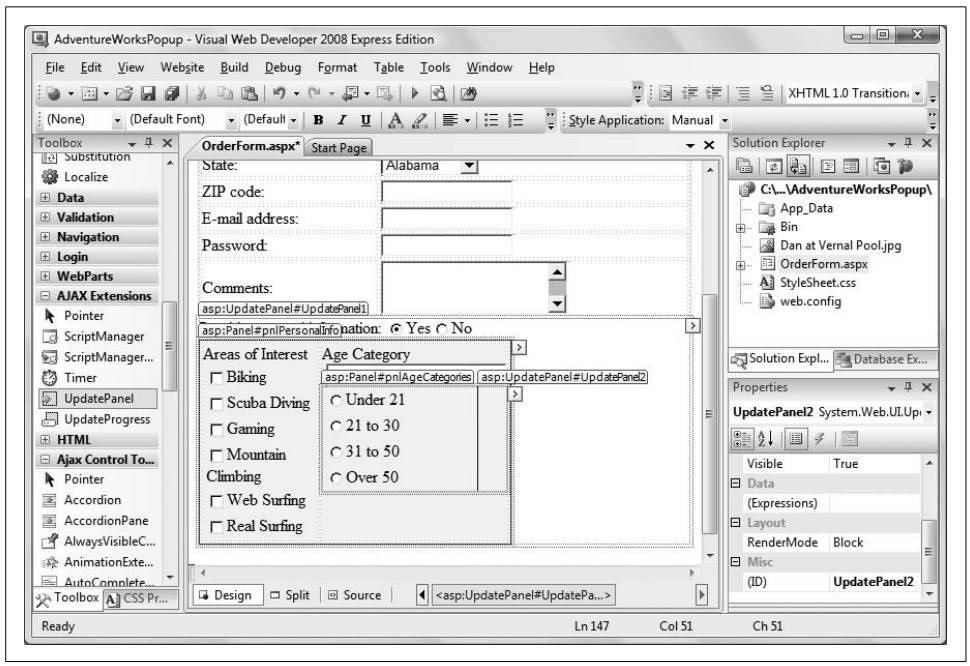


Figure 3-16. *The Design view of your form with the PopupControlExtender in place. Notice the UpdatePanel with the RadioButtonList inside it.*

The final step is to create an event handler for the `RadioButtonList` control (`rblAge`) to handle a selection change. You can do this easily, as you've seen before—double-click on `rblAge` in Design view. This will open up the code-behind file, create a skeleton event handler called `rblAge_SelectedIndexChanged`, and then place the cursor inside that method, ready for typing. Enter the following line of code:

```
txtAgeCategory_PopupControlExtender.Commit(rblAge.SelectedValue)
```

This line of code will be executed each time the user changes the selection within the `RadioButtonList`. The value the user selected is retrieved, and the `Commit` method is called to tell the `PopupControlExtender` to force the page to automatically update itself.

Looking at the Source view, the table cell containing the Age Category components should now look like the code in Example 3-6.

Example 3-6. *The PopupControlExtender control has a target control, and a separate control that pops up*

```
<td >
    Age Category<br />
    <asp:TextBox ID="txtAgeCategory" runat="server" Width="175px">
    </asp:TextBox>

    <cc1:TextBoxWatermarkExtender ID=" txtAgeCategory_TextBoxWatermarkExtender "
        runat="server" Enabled="True"
        TargetControlID="txtAgeCategory"
        WatermarkText="Click here for age categories">
    </cc1:TextBoxWatermarkExtender>

    <cc1:PopupControlExtender ID=" txtAgeCategory_PopupControlExtender "
        runat="server" Enabled="True"
        TargetControlID="txtAgeCategory"
        PopupControlID="pnlAgeCategories"
        Position="Bottom">
    </cc1:PopupControlExtender>

    <asp:Panel ID="pnlAgeCategories" runat="server" >
        <asp:UpdatePanel ID="UpdatePanel2" runat="server">
            <ContentTemplate>
                <asp:RadioButtonList ID="rblAge" runat="server"
                        AutoPostBack="True" Width="150px">
                    <asp:ListItem Value="Under 21 - Enjoy it!">
                            Under 21</asp:ListItem>
                    <asp:ListItem Value="21 to 30 - Livin' Large">
                            21 to 30</asp:ListItem>
                    <asp:ListItem Value="31 to 50 - Life Is Good">
                            31 to 50</asp:ListItem>
                    <asp:ListItem Value="Over 50 - Golden Years">
                            Over 50</asp:ListItem>
                </asp:RadioButtonList>
            </ContentTemplate>
        </asp:UpdatePanel>
    </asp:Panel>
</td>
```

Don't panic! While this looks complicated, it breaks down very simply.

All you have is a `TextBox` control (with an `ID` of `txtAgeCategory`), two extenders, and a `Panel` control. The first extender is a `TextBoxWaterMarkExtender`, and the second is a `PopupControlExtender` (their relative order is unimportant). Both extenders have a `TargetControlID` attribute and they are both set to the `ID` of the `TextBox` (`txtAgeCategory`), which makes perfect sense. The `PopupControlExtender` and the `TextBoxWaterMarkExtender` are each "extending" the behavior of the `TextBox` named `txtAgeCategory`, so they both have `txtAgeCategory` as a common target.

The TextBoxWaterMarkExtender stands on its own, but the PopupControlExtender needs a bit of help. It not only needs to know its target (who is it popping up for) but it also needs to know the ID of its PopupControl—that is, the control it will pop up when it is time to go "Pop!"

In this case, the control it is popping up is pnlAgeCategories, which is an ASP.NET Panel control, and which serves to "hold" other controls within it. The first control held within this Panel is an UpdatePanel named UpdatePanel2. As you know, anything within an UpdatePanel is updated asynchronously, so you place the RadioButtonList right into the UpdatePanel.

The RadioButtonList itself consists of a series of ListItems. A ListItem may have two very important properties (and in this case, it does)—the text to display (placed between the opening and closing brackets) and a Value property. The Value property can come in handy—it gives the programmer a way to say "what value is attached to the selected radio button?" and not necessarily get back only the text that was displayed.

You also added an event handler for the SelectedIndexChanged event of the RadioButtonList. Each time the user picks a radio button, the method that you designate (in this case rblAge_SelectedIndexChanged) will be called, allowing the control to react to the change. (You can see this from the Handles clause added to the event handler declaration in the code-behind file.) The way to react in this case is to get the value stashed away with the ListItem, and display it in the text box. The way you do this is to call the Commit method of the PopupControlExtender.

The Position property of the PopupControlExtender is set to Bottom, which places the pop-up window below the target control. The options available for the Position property are Bottom, Center, Left, Right, and Top.

Go ahead and try out your page now. You'll see that the PopupControlExtender behaves as shown previously in Figures 3-12 through 3-14.

CollapsiblePanelExtender

The CollapsiblePanelExtender control extends the Panel control, allowing it to collapse and expand. This allows you to add regions to the page which the user can collapse and expand at will. A typical use for the collapsible extender would be a product detail sheet that the user can display if interested.

In our case, we'll fill the panel with "Lorem Ipsum" text and a photo of one of the authors so that his kids will believe he really contributed to this book, however little.

Lorem Ipsum has been the printing industry's standard placeholder text for over 600 years, and it is typically called *greeking* by typographers (which is ironic since it more closely resembles Latin). The text is designed to allow the reader to ignore the words and focus on the layout, though it does have its roots in Cicero's *finibus bonorum et malorum* (The Purpose of Good and Evil).

You can hide or expand the panel at will. When the `CollapsiblePanelExtender` is collapsed, the page will look something like Figure 3-17. When it is expanded, it will look like Figure 3-18.

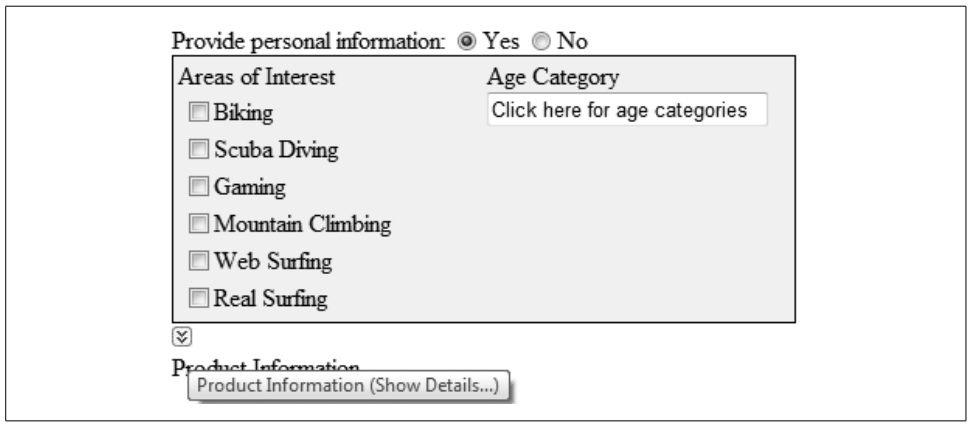


Figure 3-17. *When the CollapsiblePanelExtender is collapsed, it hides out of the way, with only the arrow indicating it's there. Note the Tooltip attached to the image.*

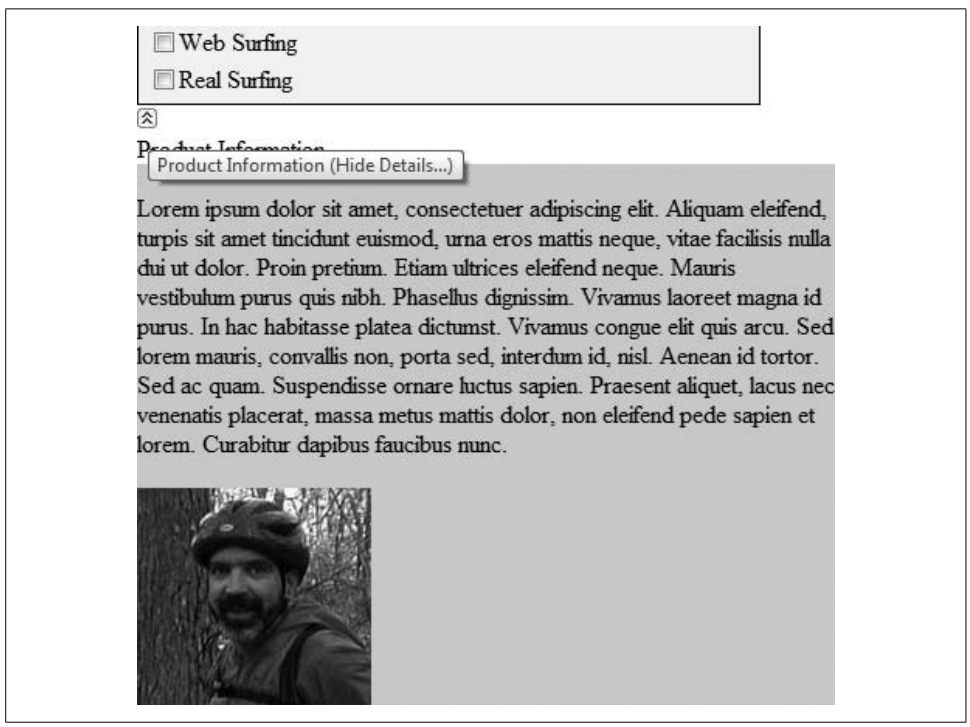

Web Surfing
Real Surfing
Product Information (Hide Details...)
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam eleifend, turpis sit amet tincidunt euismod, urna eros mattis neque, vitae facilisis nulla dui ut dolor. Proin pretium. Etiam ultrices eleifend neque. Mauris vestibulum purus quis nibh. Phasellus dignissim. Vivamus laoreet magna id purus. In hac habitasse platea dictumst. Vivamus congue elit quis arcu. Sed lorem mauris, convallis non, porta sed, interdum id, nisl. Aenean id tortor. Sed ac quam. Suspendisse ornare luctus sapien. Praesent aliquet, lacus nec venenatis placerat, massa metus mattis dolor, non eleifend pede sapien et lorem. Curabitur dapibus faucibus nunc.

Figure 3-18. *When the user clicks the arrow, the CollapsiblePanelExtender is expanded, showing the text and images. Note that the Tooltip now indicates that clicking the image will collapse the panel.*

To create this example, copy the previous web site, AdventureWorksPopupControl, to a new web site called AdventureWorksCollapsiblePanelExtender. See Appendix B for instructions on copying web sites.

Open the *OrderForm.aspx* page by double-clicking it in Solution Explorer. In Design view, drag a standard ASP.NET `Panel` control (not an AJAX `UpdatePanel`) from the Standard section of the Toolbox onto your page, below the other controls already on the page. Set the `ID` of that `Panel` to `pnlProductInfoHeader`. Within the `Panel`, you need the image for the collapse button and the text to tell the user what is inside the collapsed panel. For the graphic, drag a standard `Image` control into the `Panel`, then use the Properties window to set its `ID` to `imgProductInfo_ToggleImage` and its `ImageUrl` property to `collapse.jpg`. Type the text "Product Information" directly into the `Panel`.

The arrow graphics seen above the words "Product Information" in Figures 3-17 and 3-18 are called *collapse.jpg* and *expand.jpg*, respectively. Both images are available with the downloadable code for this book. In order to select these, or any image files directly from the Properties window, you must first add the image files to the web site by using the Website → Add Existing Item… menu item.

The markup for this `Panel`, which you can see by switching to Source view, should look something like this:

```
<asp:Panel ID="pnlProductInfoHeader" runat="server" >
   <asp:Image ID="imgProductInfo_ToggleImage"
        runat="server" ImageUrl="~/collapse.jpg" />
   Product Information
</asp:Panel>
```

It doesn't much matter in this `Image` declaration if you set the `ImageUrl` to *expand.jpg* or *collapse.jpg* because the `CollapsiblePanelExtender` control will actually be controlling which image is displayed.

Below that panel, drag a second `Panel` onto the design surface. This panel will contain the contents of the "expanded" `Panel`. Using the Properties window, set the `ID` to `pnlProductInfo`, set its `BackColor` to `LightGray` (either type that in directly or choose something similar from the color picker), and set its `Width` to `450`. From the Toolbox, drag a standard `Label` control and a standard `Image` control into this `Panel`. In either Design or Source view, set the `ID` and `Text` properties of the `Label`, along with the `ID` and `ImageUrl` properties of the `Image` control, as shown in the following code snippet:

```
<asp:Panel ID="pnlProductInfo" runat="server" BackColor="lightgray" Width="450px">
         <br />
         <asp:Label ID="myLabel" runat="server" Text="
   Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
   Aliquam eleifend, turpis sit amet tincidunt euismod, urna eros mattis
   neque, vitae facilisis nulla dui ut dolor. Proin pretium. Etiam ultrices
```

```
    eleifend neque. Mauris vestibulum purus quis nibh. Phasellus dignissim.
    Vivamus laoreet magna id purus. In hac habitasse platea dictumst. Vivamus
    congue elit quis arcu. Sed lorem mauris, convallis non, porta sed, interdum
    id, nisl. Aenean id tortor. Sed ac quam. Suspendisse ornare luctus sapien.
    Praesent aliquet, lacus nec venenatis placerat, massa metus mattis dolor,
    non eleifend pede sapien et lorem. Curabitur dapibus faucibus nunc." />
          <br />
          <br />
          <asp:Image ID="Image1" runat="server"
             ImageUrl="Dan at Vernal Pool.jpg" /><br />
</asp:Panel>
```

With the collapsed and expanded Panels in place, you can now add the AJAX control—the `CollapsiblePanelExtender`—and set its attributes. Switch to Design view, click on the `pnlProductInfo` Panel control, and then click on its Smart Tag, selecting the Add Extender menu item. Select the `CollapsiblePanelExtender` from the extenders presented. Use the default ID of `pnlProductInfo_CollapsiblePanelExtender`. Switch to Source view and set all the other properties to match the following control declaration:

```
<cc1:CollapsiblePanelExtender ID="pnlProductInfo_CollapsiblePanelExtender"
   runat="server" Enabled="True"
   CollapseControlID="pnlProductInfoHeader"
   Collapsed="true"
   CollapsedImage="expand.jpg"
   CollapsedText="Product Information (Show Details...)"
   ExpandControlID="pnlProductInfoHeader"
   ExpandedImage="collapse.jpg"
   ExpandedText="Product Information (Hide Details...)"
   ImageControlID="imgProductInfo_ToggleImage"
   SuppressPostBack="true"
   TargetControlID="pnlProductInfo">
</cc1:CollapsiblePanelExtender>
```

Here again, you must use Source view as this is another of the controls with properties that are not accessible in Design view.

The Design view of the web site will look something that shown in Figure 3-19.

The meaning of these properties is as follows:

CollapseControlID/ExpandControlID
: The controls that will expand or collapse the panel on a click, respectively. If these values are the same, as they are in this example, the panel will toggle its state with each click. Set both of these to `pnlProductInfoHeader`.

Collapsed
: Indicates the initial state of the collapsible `Panel`. For this example, set to `true`, it will start out in the collapsed state; if this is set to `false`, it will start out open. You would usually want the panel to start out collapsed.

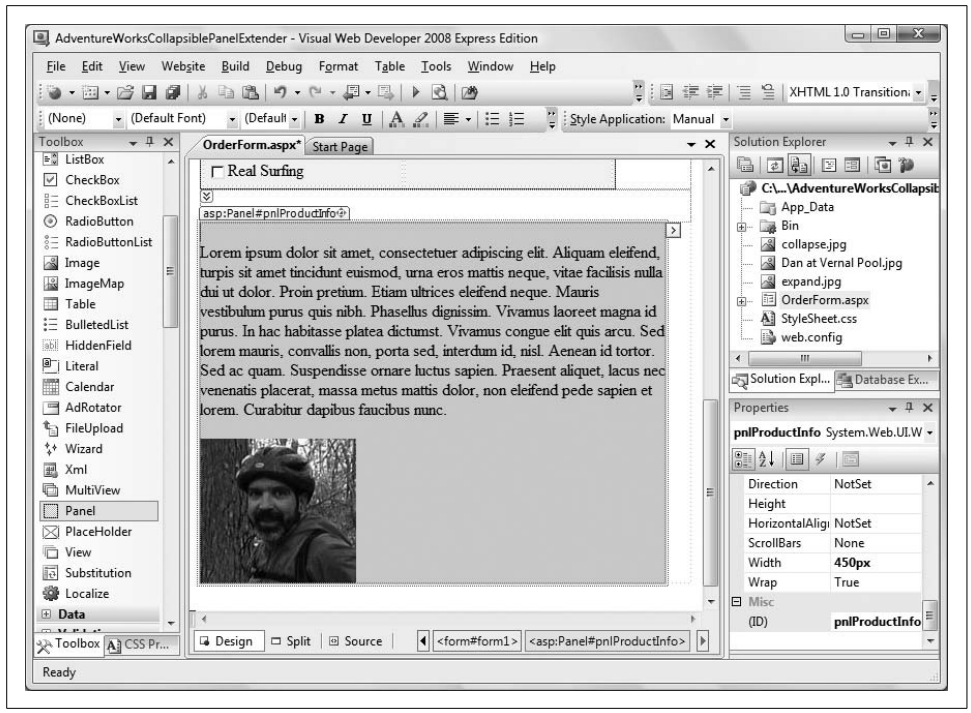


Figure 3-19. *Design view of the AdventureWorksCollapsiblePanelExtender web site showing the Panel control to be extended and the CollapsiblePanelExtender control that does the extending.*

ImageControlID

The `ID` of an `Image` control into which an icon indicating the status (collapsed or expanded) of the `Panel` will be placed. The extender will replace the source of this Image with the `CollapsedImage` and `ExpandedImage` URLs as appropriate. If the `ExpandedText` or `CollapsedText` properties are set, they are used as the alternate text for the image, also displaying as a tool tip. Set this to `imgProductInfo_ToggleImage`.

CollapsedImage

The path to an image used by `ImageControlID` when the `Panel` is collapsed. If the `Panel` is collapsed, you want readers to see an icon indicating that they can expand it. Therefore, set this property to `expand.jpg`.

CollapsedText

The text to show in the control specified by `CollapseControlID` when the `Panel` is collapsed. This text is used as the alternate text of the image if `ImageControlID` is set, also displaying as a tool tip. Set this to "Product Information (Show Details…)."

ExpandDirection

This property can be set to `Vertical` or `Horizontal` to determine whether the `Panel` expands top-to-bottom or left-to-right. For this exercise, set it to `Vertical`.

ExpandedImage
: The path to an image used by `ImageControlID` when the `Panel` is expanded. If the `Panel` is expanded, you want readers to see an icon indicating that they can collapse it. Therefore, set this to `collapse.jpg`.

ExpandedText
: The text to show in the control specified by `ExpandControlID` when the `Panel` is expanded. This text is used as the alternate text of the image if `ImageControlID` is set, also displaying as a tool tip. Set this text to "Product Information (Hide Details…)."

SuppressPostBack
: If set to `true`, ensures that the control does not cause a post back when the control is expanded or contracted. That's what you want, so set this property to `true`.

TargetControlID
: The control that will be expanded or collapsed, in this case, `pnlProductInfo`. The key thing to realize here is that the `CollapsiblePanelExtender` does not itself expand or collapse, it is used to expand and collapse a *different* control and this property (`TargetControlID`) identifies the panel it will extend.

Run the web site and you will initially see the `Panel` collapsed, as shown back in Figure 3-17. Click on the icon above Product Information and the Panel expands, displaying its information, as shown previously in Figure 3-18.

Source Code Listing

The complete source code for the final example in this chapter is shown in Example 3-7.

Example 3-7. *OrderForm.aspx*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="OrderForm.aspx.vb"
   Inherits="_Default" %>

<%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit"
   TagPrefix="cc1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>AdventureWorks</title>
   <style type="text/css">
      @import url(StyleSheet.css);
   </style>
</head>
<body>
   <form id="form1" runat="server">
   <asp:ScriptManager ID="ScriptManager1" runat="server">
```

Example 3-7. *OrderForm.aspx (continued)*

```
</asp:ScriptManager>
<div>
   <h1>
      AdventureWorks Order Form</h1>
   <table>
      <tr>
         <td>
            Customer Name:
         </td>
         <td>
            <asp:TextBox ID="txtName" runat="server"
               CssClass="unwatermarked"></asp:TextBox>
            <cc1:TextBoxWatermarkExtender ID="txtName_TextBoxWatermarkExtender"
               runat="server" Enabled="True" TargetControlID="txtName"
               WatermarkCssClass="watermarked" WatermarkText="Your name">
            </cc1:TextBoxWatermarkExtender>
         </td>
      </tr>
      <tr>
         <td>
            Address:
         </td>
         <td>
            <asp:TextBox ID="txtAddress" runat="server"
               CssClass="unwatermarked"></asp:TextBox>
            <cc1:TextBoxWatermarkExtender ID="txtAddress_TextBoxWatermarkExtender"
               runat="server" Enabled="True" TargetControlID="txtAddress"
               WatermarkCssClass="watermarked" WatermarkText="Your address">
            </cc1:TextBoxWatermarkExtender>
         </td>
      </tr>
      <tr>
         <td>
            City:
         </td>
         <td>
            <asp:TextBox ID="txtCity" runat="server"></asp:TextBox>
         </td>
      </tr>
      <tr>
         <td>
            State:
         </td>
         <td>
            <asp:DropDownList ID="ddlState" runat="server">
               <asp:ListItem Value="AL">Alabama</asp:ListItem>
               <asp:ListItem Value="AK">Alaska</asp:ListItem>
               <asp:ListItem Value="CA">California</asp:ListItem>
               <asp:ListItem Value="CT">Connecticut</asp:ListItem>
            </asp:DropDownList>
         </td>
      </tr>
```

Example 3-7. *OrderForm.aspx (continued)*

```
        <tr>
           <td>
              Zip:
           </td>
           <td>
              <asp:TextBox ID="txtZip" runat="server"></asp:TextBox>
           </td>
        </tr>
        <tr>
           <td>
              E-mail:
           </td>
           <td>
              <asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>
           </td>
        </tr>
        <tr>
           <td>
              Password:
           </td>
           <td>
              <asp:TextBox ID="txtPassword" runat="server"
                 TextMode="Password"></asp:TextBox>
           </td>
        </tr>
        <tr>
           <td>
              Comment:
           </td>
           <td>
              <asp:TextBox ID="txtComment" runat="server" Rows="3"
                 TextMode="MultiLine"></asp:TextBox>
           </td>
        </tr>
     </table>
     <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
           Provide Personal Information:
           <asp:RadioButton ID="rbYes" runat="server" AutoPostBack="true"
              Checked="true" GroupName="grpPersonalInfo" Text="Yes"
              ToolTip="Click Yes to gather personal information - no to skip that step"
              />
           <asp:RadioButton ID="rbNo" runat="server" AutoPostBack="true"
              GroupName="grpPersonalInfo" Text="No"
              ToolTip="Click Yes to gather personal information - no to skip that step"
              />
           <asp:Panel ID="pnlPersonalInfo" runat="server" BorderWidth="1px"
              Width="300px" BackColor="Beige">
              <table>
                 <tr valign="top">
                    <td>
                       Areas of Interest
```

Example 3-7. *OrderForm.aspx (continued)*

```
                        <asp:CheckBoxList ID="cblAreas" runat="server"
                              AutoPostBack="True">
                           <asp:ListItem>Biking</asp:ListItem>
                           <asp:ListItem>Scuba Diving</asp:ListItem>
                           <asp:ListItem>Gaming</asp:ListItem>
                           <asp:ListItem>Mountain Climbing</asp:ListItem>
                           <asp:ListItem>Web Surfing</asp:ListItem>
                           <asp:ListItem>Real Surfing</asp:ListItem>
                        </asp:CheckBoxList>
                     </td>
                     <td>
                        Age Category
                        <br />
                        <asp:TextBox ID="txtAgeCategory" runat="server"
                           Width="175px"></asp:TextBox>
                        <cc1:TextBoxWatermarkExtender
                           ID="txtAgeCategory_TextBoxWatermarkExtender" runat="server"
                           Enabled="True" TargetControlID="txtAgeCategory"
                           WatermarkText="Click here for age categories">
                        </cc1:TextBoxWatermarkExtender>
                        <cc1:PopupControlExtender
                           ID="txtAgeCategory_PopupControlExtender" runat="server"
                           Enabled="True" PopupControlID="pnlAgeCategories"
                           Position="Bottom" TargetControlID="txtAgeCategory">
                        </cc1:PopupControlExtender>
                        <asp:Panel ID="pnlAgeCategories" runat="server">
                           <asp:UpdatePanel ID="UpdatePanel2" runat="server">
                              <ContentTemplate>
                                 <asp:RadioButtonList ID="rblAge" runat="server"
                                       AutoPostBack="true" Width="150px">
                                    <asp:ListItem Value="Under 21 - Enjoy it!">
                                       Under 21</asp:ListItem>
                                    <asp:ListItem Value="21 to 30 - Livin' Large">
                                       21 to 30</asp:ListItem>
                                    <asp:ListItem Value="31 to 50 - Life Is Good">
                                       31 to 50</asp:ListItem>
                                    <asp:ListItem Value="Over 50 - Golden Years">
                                       Over 50</asp:ListItem>
                                 </asp:RadioButtonList>
                              </ContentTemplate>
                           </asp:UpdatePanel>
                        </asp:Panel>
                     </td>
                  </tr>
               </table>
            </asp:Panel>
         </ContentTemplate>
      </asp:UpdatePanel>
   </div>
   <asp:Panel ID="pnlProductInfoHeader" runat="server">
      <asp:Image ID="imgProductInfo_ToggleImage" runat="server"
         ImageUrl="~/collapse.jpg" />
```

Example 3-7. *OrderForm.aspx (continued)*

```
      Product Information
   </asp:Panel>
   <asp:Panel ID="pnlProductInfo" runat="server" BackColor="LightGray" Width="450px">
      <br />
      <asp:Label ID="myLabel" runat="server" Text="
         Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
         Aliquam eleifend, turpis sit amet tincidunt euismod, urna eros mattis
         neque, vitae facilisis nulla dui ut dolor. Proin pretium. Etiam ultrices
         eleifend neque. Mauris vestibulum purus quis nibh. Phasellus dignissim.
         Vivamus laoreet magna id purus. In hac habitasse platea dictumst. Vivamus
         congue elit quis arcu. Sed lorem mauris, convallis non, porta sed, interdum
         id, nisl. Aenean id tortor. Sed ac quam. Suspendisse ornare luctus sapien.
         Praesent aliquet, lacus nec venenatis placerat, massa metus mattis dolor,
         non eleifend pede sapien et lorem. Curabitur dapibus faucibus nunc." />
      <br />
      <br />
      <asp:Image ID="Image1" runat="server" ImageUrl="Dan at Vernal Pool.jpg" />
      <br />
   </asp:Panel>
   <cc1:CollapsiblePanelExtender ID="pnlProductInfo_CollapsiblePanelExtender"
      runat="server" Enabled="True"
      CollapseControlID="pnlProductInfoHeader" Collapsed="true"
      CollapsedImage="expand.jpg" CollapsedText="Product Information (Show Details...)"
      ExpandControlID="pnlProductInfoHeader"
      ExpandedImage="collapse.jpg" ExpandedText="Product Information (Hide Details...)"
      ImageControlID="imgProductInfo_ToggleImage"
      SuppressPostBack="true" TargetControlID="pnlProductInfo">
   </cc1:CollapsiblePanelExtender>
   </form>
</body>
</html>
```

The code-behind for this page is contained in Example 3-8.

Example 3-8. *OrderForm.aspx.vb*

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub YesNoEventHandler(ByVal sender As Object, _
     ByVal e As System.EventArgs) _
     Handles rbNo.CheckedChanged, rbYes.CheckedChanged

        pnlPersonalInfo.Visible = rbYes.Checked
    End Sub

    Protected Sub rblAge_SelectedIndexChanged(ByVal sender As Object, _
             ByVal e As System.EventArgs) _
             Handles rblAge.SelectedIndexChanged
          txtAgeCategory_PopupControlExtender.Commit(rblAge.SelectedValue)
    End Sub
End Class
```

Summary

- AJAX is a technique for shifting much of the processing off the server and to the user's browser, which dramatically increases performance, both actual and perceived.
- The ASP.NET AJAX control library contains a number of controls that can be used just as easily as standard ASP.NET controls, meaning you don't need to know the JavaScript that makes the controls work.
- The `ScriptManager` control is the key control that makes ASP.NET AJAX possible by managing the JavaScript for you behind the scenes. The control is placed on every AJAX-enabled page by default, and its `EnablePartialRendering` property is set to `True`, so you don't need to do anything yourself.
- Placing controls inside an `UpdatePanel` control enables you to update those controls without posting back to the server.
- AJAX asynchronous postbacks are not added to the browser's history, which can cause unexpected behaviors of page controls when the Forward and Back browser buttons are used. With ASP.NET 3.5 Service Pack 1, you can create history points that allow the Forward and Back buttons of the user's browser to function as expected.
- The AJAX Control Toolkit, which is a separate download, has a number of controls called extenders that enhance existing controls, rather than having stand-alone functionality themselves. Extender controls have a `TargetControlID` property that you use to set the existing control that the extender is extending.
- The `TextBoxWaterMarkExtender` adds a watermark effect to an existing textbox, providing a prompt for the reader to enter data.
- `TextBoxWaterMarkExtender` can apply a separate style to a text box, if you have a style sheet defined for the project, or it can just add the text you specify.
- The `PopupControlExtender` can help you make efficient use of the space on your page, by hiding some content until the user clicks on a control.
- You can apply the `CollapsiblePanelExtender` to a regular `Panel` control, causing it to hide most of its content until the user clicks on it. The `Panel` then expands, displaying its content, until the user collapses it.

Now you have a good handle on the basic controls, and you've also seen how to use some of the AJAX extenders to apply some really clever effects to them. The AdventureWorks order form you've been progressively building is becoming pretty sophisticated by now. As we've mentioned, though, it doesn't connect to anything behind the scenes, so users can't see the AdventureWorks products, and can't yet place their orders. To do that, you need to learn how to interact with a database. ASP.NET provides a number of controls for retrieving data from a database and displaying it in a number of different ways. With AJAX, they get even better. You'll learn all about them in the next chapter.

Quiz

1. What do you need to do to use AJAX-enabled controls on your site?
2. Which property of the `ScriptManager` control enables asynchronous postback?
3. What control do you need to place on your page to enable asynchronous updates?
4. Which event do you need to handle on the `ScriptManager` control to modify the behavior of the browser Forward and Back buttons for AJAX controls?
5. Can you make a page that contains only an extender control?
6. What property is common to all the AJAX extender controls?
7. What view do you use to set the `WatermarkText` property?
8. What's the advantage of using the `PopupControlExtender`?
9. What method of the `PopupControlExtender` do you need to call to display the results?
10. What control does the `CollapsiblePanelExtender` work with?

Exercises

Exercise 3-1. We'll start things off simply. Suppose you have a store that ships only to certain states in the northeastern United States. In your order form, you want to restrict users to only those states as their shipping destination, so you want to use a drop-down list. You also want to save space on your form, though, so you want to hide that delivery list in a `Panel` with a `PopupControlExtender`. For this exercise, you'll only produce the part of the form where users would enter the shipping State. The finished form should look like Figure 3-20.

The drop-down list should contain just the six states shown in Figure 3-20. When the user chooses one of the states, the state's two-letter postal abbreviation should appear in the text box.

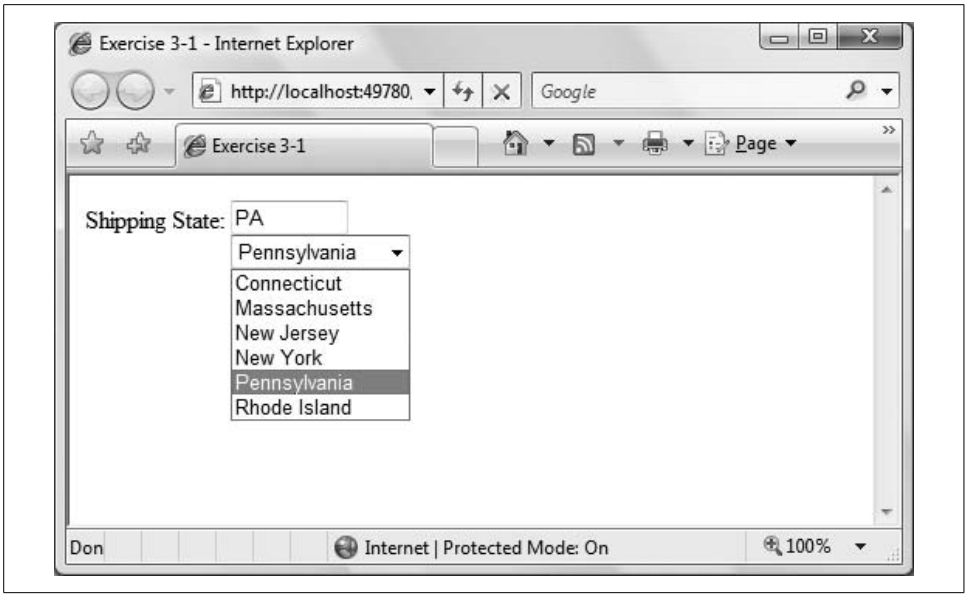


Figure 3-20. *Your goal for Exercise 3-1.*

Exercise 3-2. Most of the AJAX control extenders that we've shown you in this chapter just do one thing, although they do it very well. There are, however, many more extenders that we haven't shown you, and more are being added all the time. Each one is different, and covering them all in detail would require more space than we have—or would be out of date almost immediately. The best way to learn about the AJAX control extenders is to go to the ASP.NET AJAX Control Toolkit page at *http://ajax.asp.net/ajaxtoolkit/*. There you'll find the latest extenders with examples of how to use them. Many of the extenders are fairly simple, and have properties you can set easily. In this exercise, you'll need to use that documentation as you try out a new extender.

For this exercise, you'll use the `RoundedCorners` extender. Create a new page that includes a `Panel` with the dimensions 150 pixels × 100 pixels, colored light gray. The `Panel` should contain a `Label`, 50 pixels wide, colored dark gray, with text that's white and bold (feel free to use livelier colors; we're choosing ones here that will show up in the printed book). The `Panel` should have only its top corners rounded to a radius of 8. The `Label` should have all its corners rounded to a radius of 2. You'll need the documentation to tell you how to do all that. The result should look like Figure 3-21.

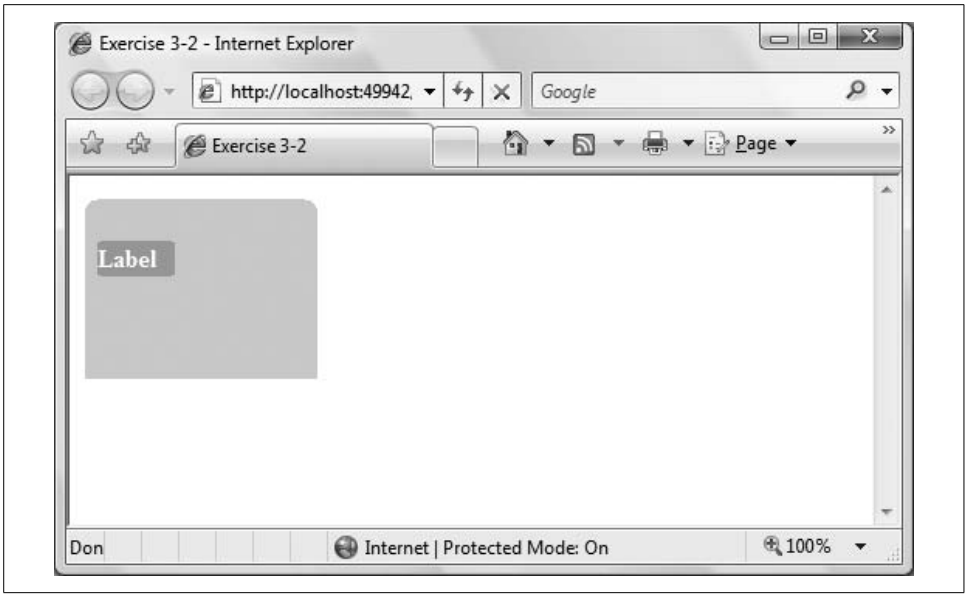


Figure 3-21. *Your goal for Exercise 3-2.*

Exercise 3-3. The `SliderExtender` is another interesting extender, but slightly more complicated than it looks. Create a new page that uses the slider extender to simulate a volume control. The slider should be set horizontally, and should have a range from 0 to 10. The results should look like Figure 3-22. (Hint: the documentation is somewhat unclear. You'll need two `TextBoxes`—the slider prevents one from displaying, so you need to use a second `TextBox`, called a *bound control*, to see the value the slider is set to.)

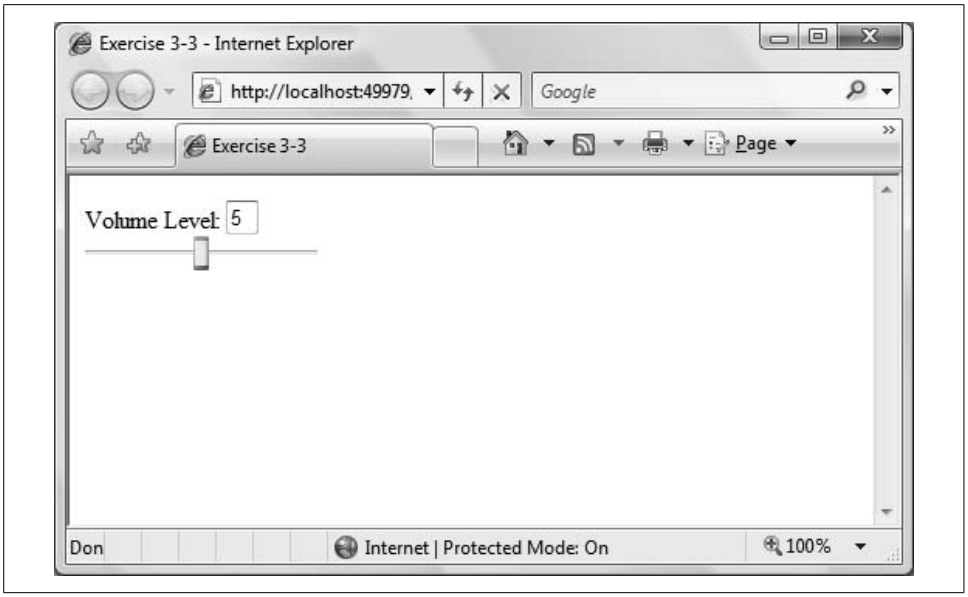


Figure 3-22. *Your goal for Exercise 3-3.*

Exercise 3-4. For this exercise, you'll need to use Visual Web Developer (or Visual Studio 2008) with Service Pack 1. If you don't have that version, this exercise won't work. Create a website that contains a `ScriptManager` and an `UpdatePanel`. Inside the UpdatePanel should be a single `TextBox` and a `Button`. Use the technique for controlling the browser history so that you can enter text in the textbox, save it to the history, change the text, and then use the Forward and Back buttons to restore the previously entered text. The result should look like Figure 3-23. (Note that in Figure 3-23, we've moved the controls down on the page so that the history menu won't cover them. You don't need to do that, though.)

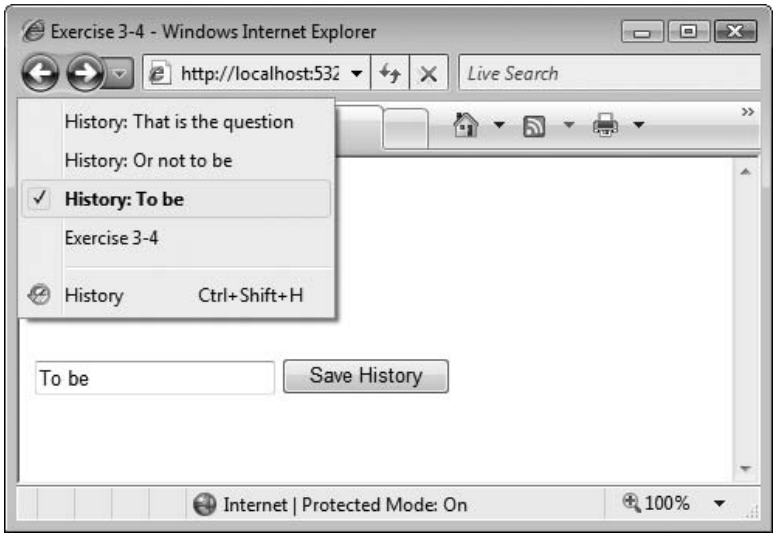


Figure 3-23. *Your goal for Exercise 3-4.*

CHAPTER 4

Saving and Retrieving Data

So far, you've seen how to make functional web pages with clever and useful controls. You know how to change the appearance of the page in response to user selections, and how to use AJAX to enhance the performance of your application. But the applications you've made so far have been limited in what they can actually *do*. In this chapter, we add the most frequently sought-after functionality: the ability to retrieve, display, change, and store data.

Think about the web sites you visit most often, and you'll find that almost all of them have one thing in common—they interact with persistent data. *Persistent data* is data that survives a single session; data that you expect will be there the next time you visit. In fact, it may even be data that can have significant financial consequences.

Shopping sites have databases to track their inventories and customer transactions. News sites keep databases with articles and photos in them, perhaps referenced by topic and date. Search engines use unimaginably large (and wicked-fast) databases.

Nearly every real-world commercial web application must be able to perform the four essential "CRUD" interactions with a database: Create, Read, Update, and Delete.

Fortunately, ASP.NET provides controls that make it easy and fast to perform these essential activities. We will demonstrate these interactions with SQL Server Express Edition (or its big brother, SQL Server) but they work equally well—or nearly so—with Microsoft Access and most commercial databases. In theory, you can interact with virtually any collection of data or with XML files, but that is an advanced topic we won't go into here.

Along the way, we'll show you enough about database interactions that even if you've never used a relational database, such as SQL Server Express, you'll have little trouble working with one through your web application.

Getting Data from a Database

To see how to interact with a database, you'll begin by creating a web application that you can use to display information about the AdventureWorks database. You'll start out by simply retrieving and displaying a selection of data. These exercises will teach you how to connect your controls to a database, to retrieve, filter, and sort the data, and then use the myriad options for presenting it attractively.

As you may remember, AdventureWorks is a free database from Microsoft that represents a fictional company that sells outdoor and extreme sports gear. The database tracks products, inventory, customers, transactions, and suppliers.

See Appendix A for instructions on installing this sample database if you have not already done so.

ASP.NET includes a number of controls specifically designed for displaying data. We'll focus on the `GridView` control, but other data controls include the `ListView`, `DataList`, `Repeater`, `DetailsView`, and `FormView`. The `GridView` control displays columns and rows of data and allows sorting and paging. It is by far the most popular data display control, and is ideal for understanding how data display controls interact with data-retrieval controls and code. The `GridView` control allows the user to click on a column header to sort the data. `GridViews` also let you present just a small subset of the data at one time, called a *page*, with links for easy access to other pages—this process is called "paging" through data. You can do these, and for numerous other data manipulations, with very little programming. A `GridView` with data from the AdventureWorks database is shown in Figure 4-1.

The other data controls mentioned here are used in scenarios where a straight, tabular display of data is not what you are looking for.

The `DetailsView` control is used for displaying the data one record at a time. The `FormView` is used to for displaying a single record in a master/detail relationship. The `ListView` (which is new in version 3.5 of the .NET Framework), `DataList` and `Repeater` controls offer the maximum flexibility for displaying data. These three controls have no inherent look of their own, but require that you, the developer, construct *templates* that describe how the data will appear.

The `ListView`, in particular, offers all the features of the `GridView`, including paging, sorting, data update, and so on, with unlimited display flexibility. You can use it to display repeating data in a nontabular format or to group records according to some criteria.

For complete coverage of these other data controls, see our more advanced book, *Programming ASP.NET,* Fourth Edition (O'Reilly).

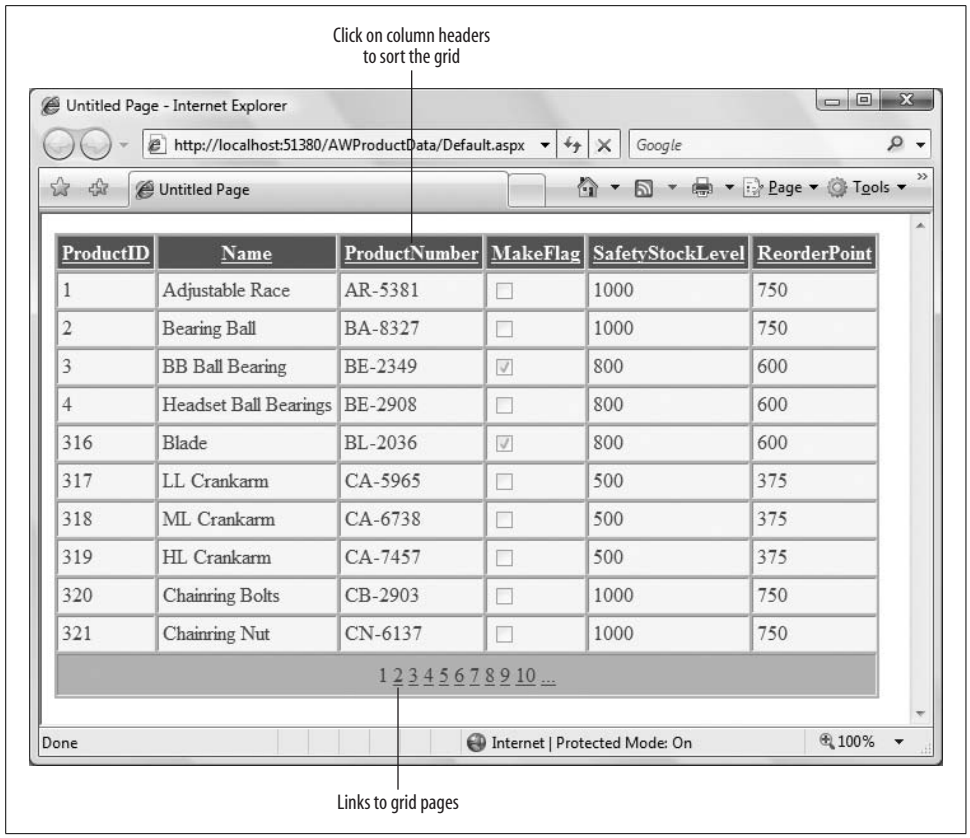


Figure 4-1. *This GridView control displays data from the AdventureWorks database in a table format that makes it easier to read, and allows users to click the column headings to sort the data.*

Binding Data Controls

Database information is stored in memory as *tables* (just as it is retrieved from a relational database). Tables consist of rows and columns that match nicely to the `GridView` control.

You could write code to pick out each piece of data you want and write it into the appropriate row or column of the data control, but that's time-consuming and error-prone. It's more efficient and safer to *bind* the control directly to the underlying data.

In the early days of Graphical User Interface (GUI) programming, binding was a bit of a "trick"—great for simple programs, but useless for commercial applications because the minute you wanted to do anything out of the ordinary, the binding would become a straitjacket. Microsoft has solved that in ASP.NET by exposing events on the data controls that allow you to insert custom code at every stage of the retrieval and binding of the data to the control.

SQL CHEAT SHEET

Relational Database Concepts

Within a relational database, data is stored in *tables*, and those tables consist of *rows* and *columns*, an arrangement that's easy to visualize. You might have columns for customer numbers, names, and addresses, for example. Each column represents a different type of data, and each row represents an individual data *record*. In this example, a row would have the customer number, name, and address of an individual customer.

Within a given table, you generally need a way to uniquely identify a specific row, so that you can retrieve just that row and no other. That's called a *primary key* or a *unique key*. In this example, you might designate the customer number to be a unique key.

The thing that makes a database relational is its use of *foreign keys*. For example, you might have a table of customer transactions named Orders, listing the items a customer ordered and when. You could store the customer's shipping information in that table as well, but if Mr. Smith orders from you a dozen times every month, you'd have his shipping information stored in the table many times over, which is wasteful. Instead, you could just store the customer number as a column in your Orders table, and keep Mr. Smith's address information in a separate, but *related* table, where you'd only have to store it once. The customer number in this case is the foreign key in the Orders table—it defines the relationship between the Orders table and the Customers table.

To read the data from a relational database, you use a *query* to define the data that you want to retrieve. In plain English, you'd say something like, "Get me the names of all the customers who live in Philadelphia." Of course, you need a formal way of relaying that instruction to the database, and that's where SQL comes in. SQL (pronounced "sequel") stands for "Structured Query Language," and gives you a way to define the queries. In SQL, the query we just mentioned might look like this:

```
SELECT Name FROM Customers WHERE City = "Philadelphia"
```

There's a lot more to SQL than that, enough to fill books on its own, and you'll see some more as you go through the chapter.

Binding is most often used with the larger data controls such as `GridView` and `ListView`, but you can also bind many other controls, such as `DropDownList`, `ListBox`, `CheckBoxList`, and `RadioButtonList`. All of these controls have a `DataSource` property that identifies the source to which the control is bound. For example, you might keep a list of all your customers' names in a database. Binding that data to a `ListBox` can be a convenient way to allow a customer service representative to quickly pick a customer rather than typing in a name that might otherwise be difficult to spell.

To see how all this works, you'll build the `GridView` from Figure 4-1. Once you have it up and running, you'll add some features to it, including the ability to use the grid to update the database with new data. Later in the chapter, you will also build an example using the `ListView` control.

Create a Sample Web Page

To begin, create a new web site named *AWProductData*.

Drag the all-important `ScriptManager` control from the AJAX Extensions section of the Toolbox onto your page. Next, open the Data tab of the Toolbox. You'll find two types of objects: display controls, which are designed to present data, and `DataSource` controls, which are designed to manage interacting with data, as shown in Figure 4-2.

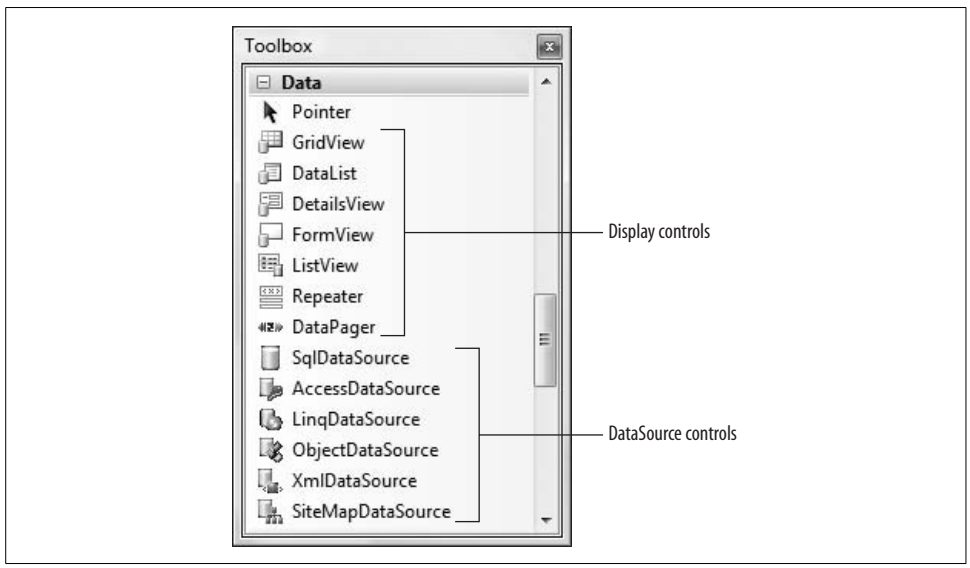


Figure 4-2. *The Data tab in the Toolbox contains the controls that you'll need to display data, and to interact with data sources.*

Using a DataSource Control

By default, the Data controls are arranged in the Toolbox so the display controls are on top, and the DataSource controls are below (You can drag them into any order you like or arrange them alphabetically by right-clicking on any control and selecting Sort Items Alphabetically.) There is a `DataSource` control for use with Microsoft SQL Server or SQL Server Express, one for Microsoft Access, one for using LINQ (Language Integrated Query, which is covered at the end of this chapter), one for any type of Object, one for use with SiteMaps (for binding to menu controls—more on this in Chapter 6), and one for XML documents as a data source.

Because the AdventureWorks database is a SQL Server database, you'll use the `SqlDataSource` control, whether you are using SQL Server or SQL Server Express. This control will allow you to access the AdventureWorks database, but first you need to direct the control where to find it.

Switch to Design view and drag the `SqlDataSource` control from the Toolbox directly onto the design surface. A Smart Tag will open, as seen in Figure 4-3.

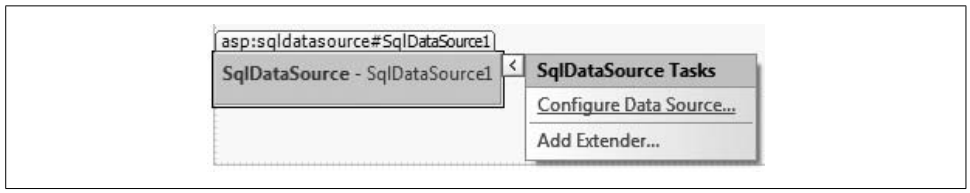


Figure 4-3. *A Smart Tag opens when you drag the SqlDataSource control onto your page, allowing you to configure the data source.*

When you click on Configure Data Source, you invoke a wizard that will walk you through the steps of configuring your data source—hooking up the control to the underlying data table(s).

The first step is to create (or choose) a data connection as seen in Figure 4-4.

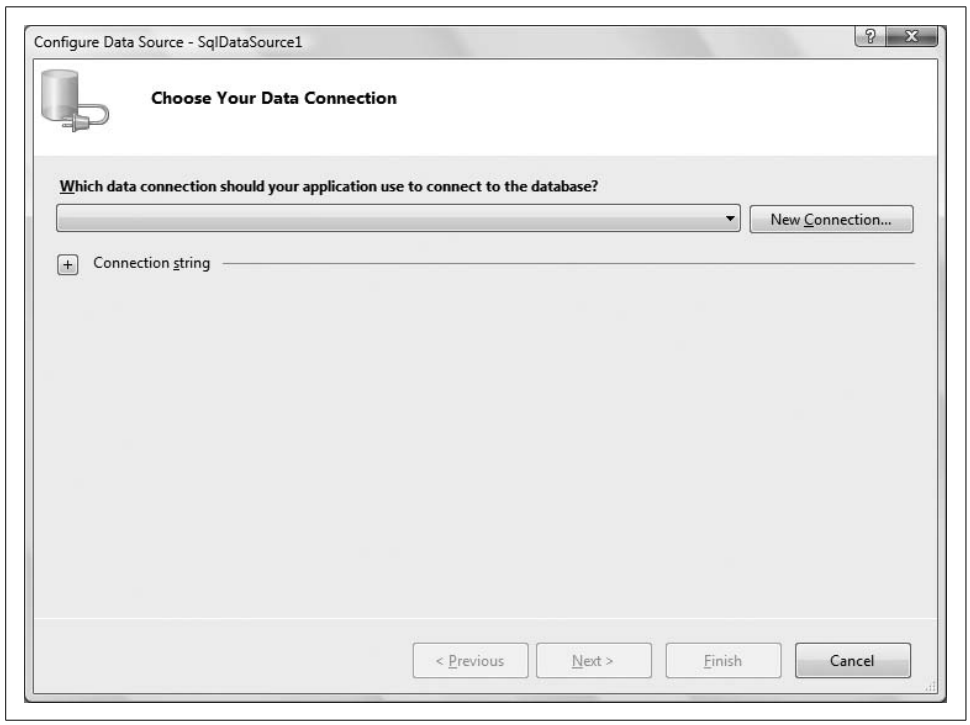


Figure 4-4. *To configure your DataSource control, you need to provide it with a data connection. You can choose a preexisting connection from the list (if you have previously created any for this web site), or create a new data connection by clicking the New Connection button.*

Any data connections that you've made before, for this web site or any other, will appear in the drop-down list. Most likely, though, you haven't created any data connections yet, so the list will be empty. To make a new connection, click the New Connection… button to get the Add connection dialog shown in Figure 4-5.

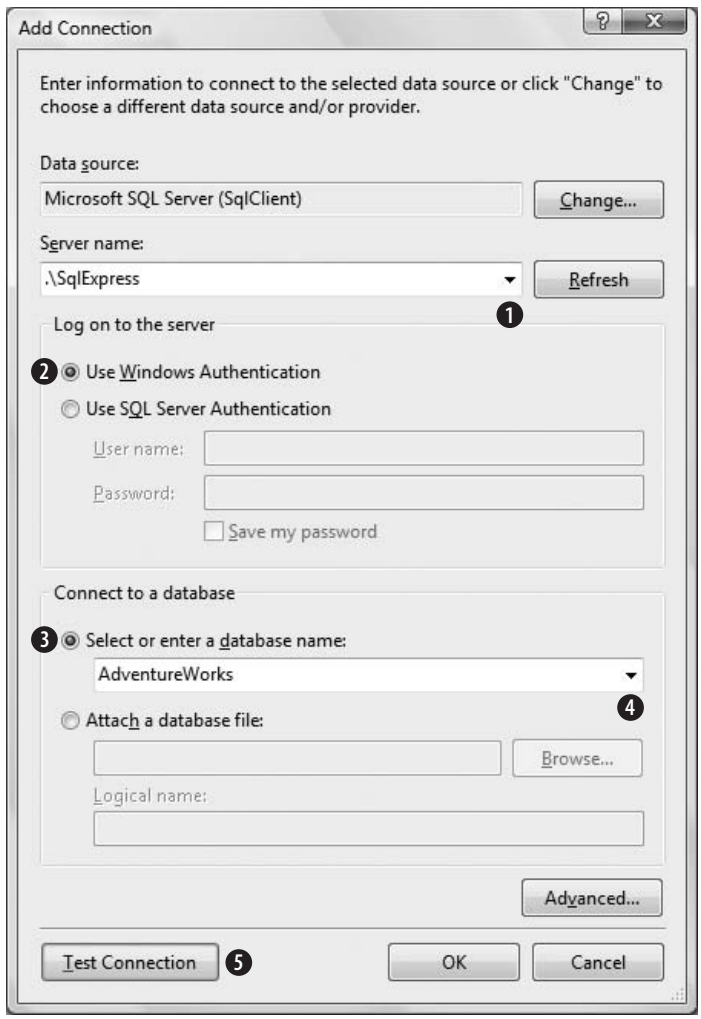


Figure 4-5. *The Add Connection dialog is where you specify a new connection for your data source. Select the server, the logon credentials, and finally the database you want to use.*

Following the steps in Figure 4-5, prepare your connection to the database:

1. Select your server from the Server Name drop-down menu. If it is not there, type the name of the server. Typically, if you are using SQLExpress, the name will be

".\SqlExpress" (dot, slash, then SqlExpress), and if you are using SQL Server, it will be the name of your computer, or it will be (local)—including the parentheses.

2. Leave the radio button set to "Use Windows Authentication."

If Windows Authentication does not work, you may need to use SQL Server authentication. If so, your database administrator will tell you what credentials to enter. They may or may not be the same as your Windows login credentials.

3. Select the option, "Select or enter a database name:".
4. Choose the AdventureWorks database in the database name drop-down.
5. Click the Test Connection button to verify that it all works.

This dialog box constructs a *connection string*, which provides the information necessary to connect to a database.

Click OK to complete the connection string and return to the Configure Data Source Wizard. Click the plus mark next to "Connection string" to see the connection string you've just created, as shown in Figure 4-6. The segment `IntegratedSecurity=True` was created when you chose Windows Authentication rather than SQL Server Authentication.

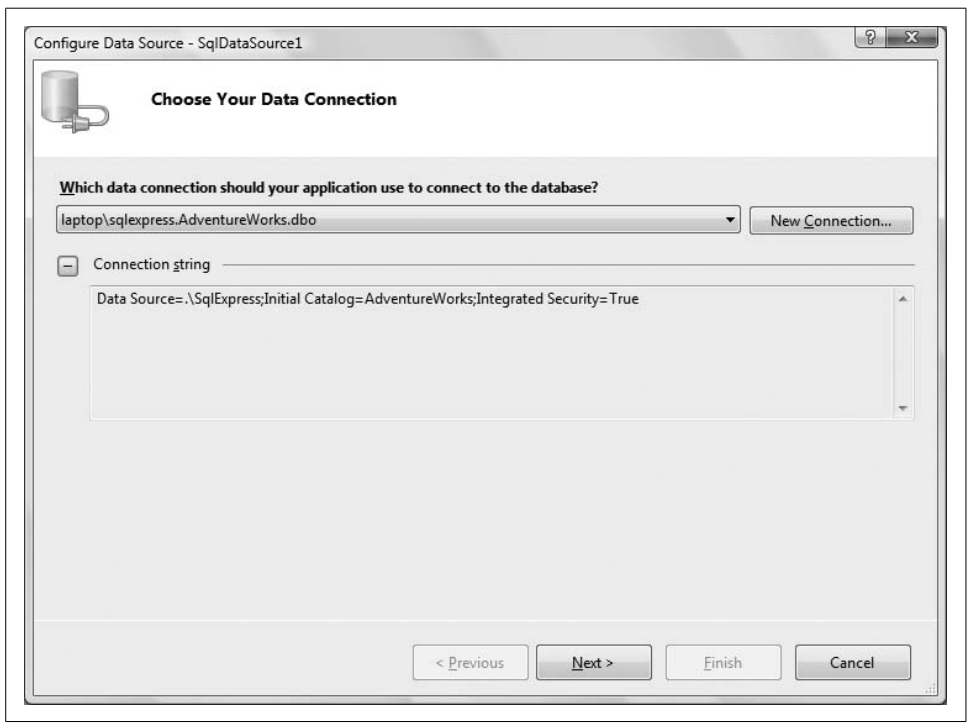


Figure 4-6. *Click the plus sign to view the connection string you just created. This is what gives your control access to the database.*

In Figure 4-6, the Wizard displays an expanded data connection in the drop-down menu, consisting of the name of the server (in this case, the local machine, `laptop`, concatenated with `sqlexpress`, followed by the name of the database and database owner). You don't need to enter this information yourself; it is created for you.

When you click Next, the wizard will ask if you'd like to save this connection string in the "application configuration file." The configuration file is an XML file that stores various settings for the web site. In an ASP.NET web site, the application configuration file is *web.config*, and saving the connection string there is an excellent idea; that way, all the pages of your web site will be able to use the connection. Make sure that the checkbox is checked, and give the string a name you can easily remember. The Wizard will make a suggestion for the name of the connection string, as shown in Figure 4-7.

Configure Data Source - SqlDataSource1
Save the Connection String to the Application Configuration File
Storing connection strings in the application configuration file simplifies maintenance and deployment. To save the connection string in application configuration file, enter a name in the text box and then click Next. If you choose not to do this, the connection string is saved in the page as a property of the data source control.
Do you want to save the connection in the application configuration file?
Yes, save this connection as:
AdventureWorksConnectionString
< Previous
Next >
Finish
Cancel

Figure 4-7. *It's a good idea to save the connection string in the application's web.config file, so you can use it again with other controls.*

This will cause the following lines to be written to the *web.config* file:

```
<connectionStrings>
   <add name="AdventureWorksConnectionString"
       connectionString="Data Source=.\SqlExpress;
          Initial Catalog=AdventureWorks;Integrated Security=True"
```

```
            providerName="System.Data.SqlClient"/>
    </connectionStrings>
```

The Wizard next prompts you to configure the SELECT statement. The SELECT statement is the SQL code the control uses to retrieve the exact subset of data you are looking for from the database. Fortunately, if you are not fluent in SQL, the Wizard will help you build the statement.

Starting with the radio buttons at the top of the dialog box, select "Specify columns from a table or view." (You would select the other button if you had a custom SQL statement prepared, as you'll see shortly.)

Selecting that radio button displays the table drop-down menu. Here, you are presented with the various tables that represent the different sets of data in the database. For this exercise, choose the Product table. The various columns from the Product table will be displayed, as shown in Figure 4-8.

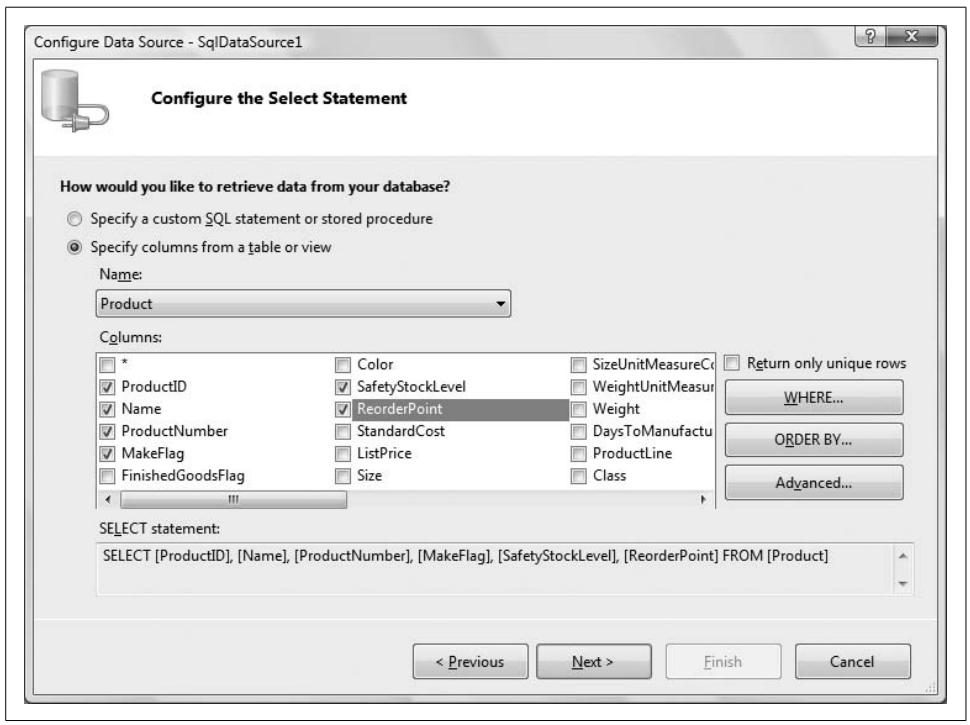


Figure 4-8. *To configure the SELECT statement, specify the table and columns within it you want to retrieve, and the Wizard builds the proper SQL statement for you…more or less.*

Simply check the columns you want retrieved, and they'll be added to the SELECT statement. The choices you make will be displayed in the text box at the bottom of the dialog. For this exercise, select the ProductID, Name, ProductNumber, MakeFlag, SafetyStockLevel, and ReorderPoint columns (you may have to scroll to the

right to see them all). You could narrow the set of data with the WHERE button, or specify the sort order in which to retrieve the data with the ORDER BY button. For the moment, you can ignore them both.

Pay No Attention to That Man Behind the Curtain

When you've completed the table setup, click Next to move to the last page of the Wizard, and then click the Test Query button. The test fails, as shown in Figure 4-9.

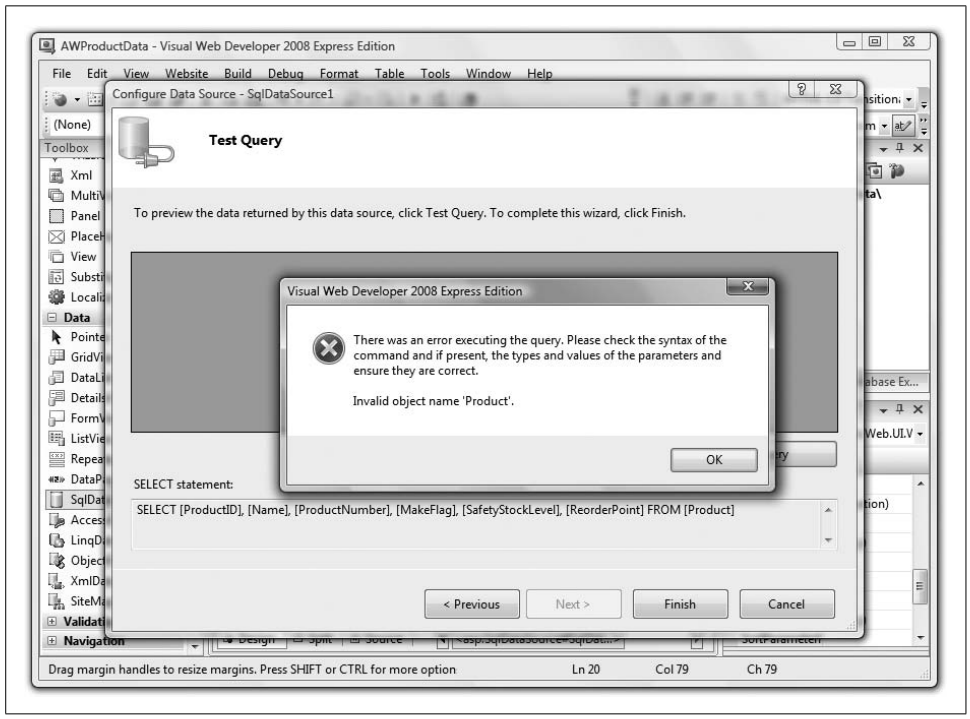

AWProductData - Visual Web Developer 2008 Express Edition
File Edit View Website Build Debug Format Table Tools Window Help
Configure Data Source - SqlDataSource1
Test Query
To preview the data returned by this data source, click Test Query. To complete this wizard, click Finish.
Visual Web Developer 2008 Express Edition
There was an error executing the query. Please check the syntax of the command and if present, the types and values of the parameters and ensure they are correct.
Invalid object name 'Product'.
OK
SELECT statement:
SELECT [ProductID], [Name], [ProductNumber], [MakeFlag], [SafetyStockLevel], [ReorderPoint] FROM [Product]
< Previous
Next >
Finish
Cancel
Toolbox
Drag margin handles to resize margins. Press SHIFT or CTRL for more option
Ln 20
Col 79
Ch 79

Figure 4-9. *The Wizard let you down; the query test failed and you're looking at this error message because this database requires a schema name in front of the table names.*

In this instance, the Wizard falls on its face. It turns out that the AdventureWorks database prefixes a *schema* name in front of each table name and the Wizard is unprepared for that. It generates a `SELECT` statement without schema names, as you saw in Figure 4-8.

Schema in this context refers to an optional name used for organizing the tables in a large database. For example, in the AdventureWorks database, all the tables relating to the HR department have the schema name `HumanResources` prefixed to every table name, separated by a period, such as `HumanResources.EmployeeAddress`. Other schemas in the AdventureWorks database include `Person`, `Production`, `Purchasing`, and `Sales`.

As mentioned, a schema name is optional in SQL. In our experience, they are rarely used, and in fact, the Wizard is unaware of them. However, the AdventureWorks database (which ships as part of Microsoft SQL Server) does use them, the Wizard becomes confused and flies off to Kansas leaving you on your own.

The square brackets surrounding each field and table name in the generated `SELECT` statement are not required, but are used to guarantee that there will be no problems if the name includes any space characters (usually a very bad idea in any case). We often remove them from the finished statement to enhance readability.

Think of this as proof that people are not yet entirely replaceable by automation. Hit the Previous button to go back one step and fix the `SELECT` statement manually. Click the radio button captioned "Specify a custom SQL statement or stored procedure," and then click Next. In the SQL Statement box, shown in Figure 4-10, type in:

```
SELECT ProductID, Name, ProductNumber, MakeFlag, SafetyStockLevel, ReorderPoint
FROM Production.Product
```

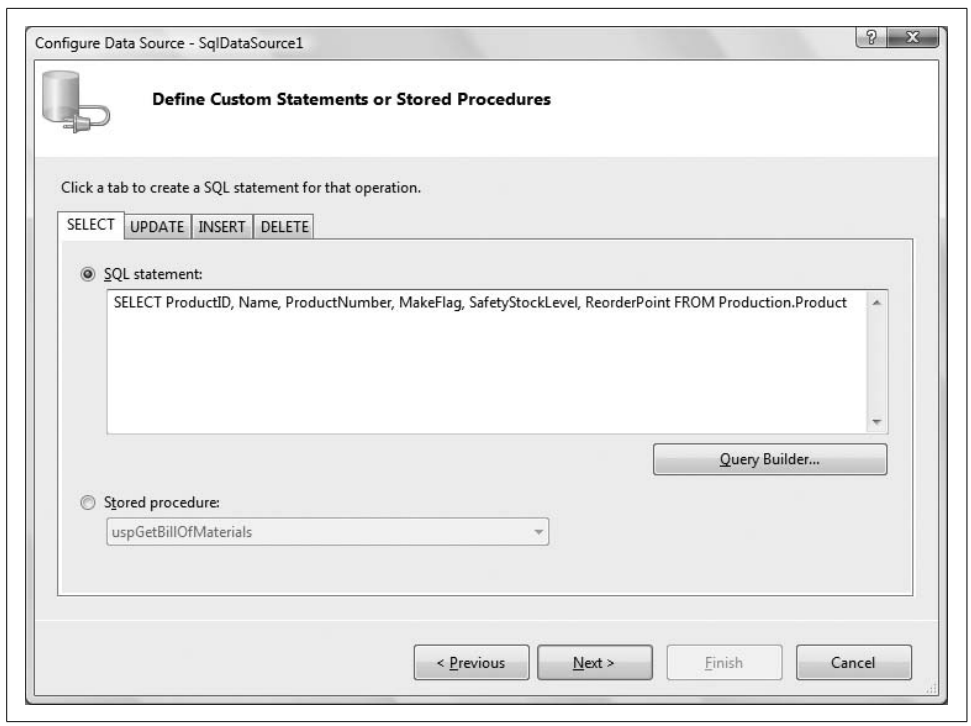


Figure 4-10. *The SQL statement editing dialog, after adding the schema name to the table name, and removing all the extraneous square brackets.*

As you can see, this is nearly the same `SELECT` statement that you built with the Wizard in Figure 4-8, except the Product table now has the required schema (`Production`) in front of it. We've also left out the square brackets on the columns, as mentioned in the note above.

Click Next to proceed to the next page of the Wizard, and then click Test Query. This time, you should get the results shown in Figure 4-11.

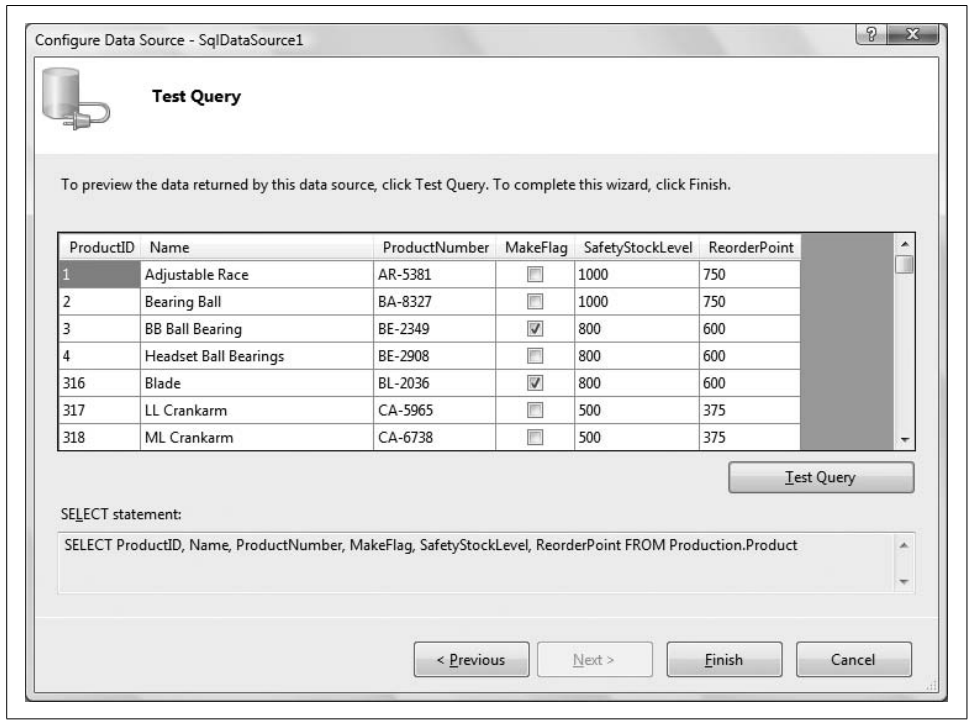

Configure Data Source - SqlDataSource1
Test Query
To preview the data returned by this data source, click Test Query. To complete this wizard, click Finish.
ProductID Name ProductNumber MakeFlag SafetyStockLevel ReorderPoint
1 Adjustable Race AR-5381 1000 750
2 Bearing Ball BA-8327 1000 750
3 BB Ball Bearing BE-2349 800 600
4 Headset Ball Bearings BE-2908 800 600
316 Blade BL-2036 800 600
317 LL Crankarm CA-5965 500 375
318 ML Crankarm CA-6738 500 375
Test Query
SELECT statement:
SELECT ProductID, Name, ProductNumber, MakeFlag, SafetyStockLevel, ReorderPoint FROM Production.Product
< Previous
Next >
Finish
Cancel

Figure 4-11. *When you test the SELECT statement this time, you'll see the results you were looking for.*

Behold—the triumph of 3 billion years of random mutation and natural selection over 50 years of automation!

Click Finish to save your work. It may not look like much, but you've just enabled your application to access the AdventureWorks database, meaning all that data is now under your control.

GridView Control

Now that the DataSource control is providing the data you want, you need a way to display it. From the Data section of the Toolbox, drag a `GridView` control onto the page. The `GridView` control recognizes automatically that a `SqlDataSource` is on the page.

If you had dragged the `GridView` onto the page first, it would have given you the opportunity to create a `SqlDataSource` rather than assuming you'd like to use one already in existence. It pretty much amounts to the same thing.

Click on the Smart Tag of the `GridView` (if it is not already open). Click the drop-down menu next to "Choose Data Source" and select the `DataSource` control you just created, as shown in Figure 4-12.

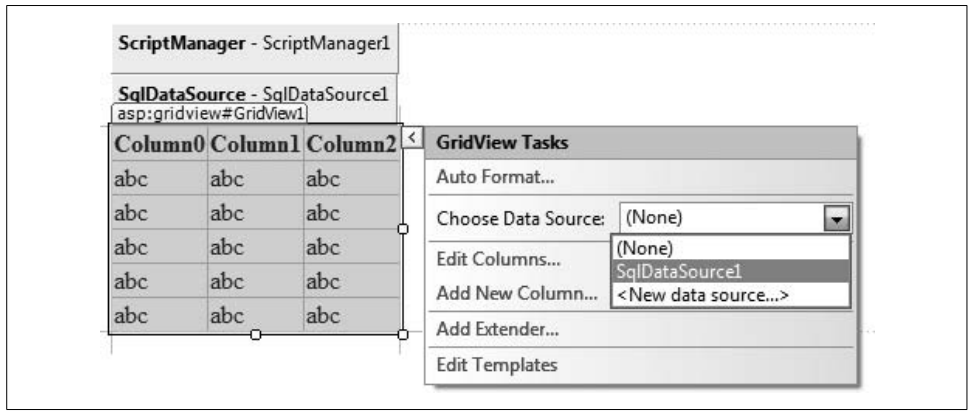


Figure 4-12. *The Smart Tag of the GridView control lets you select the data source you want to use.*

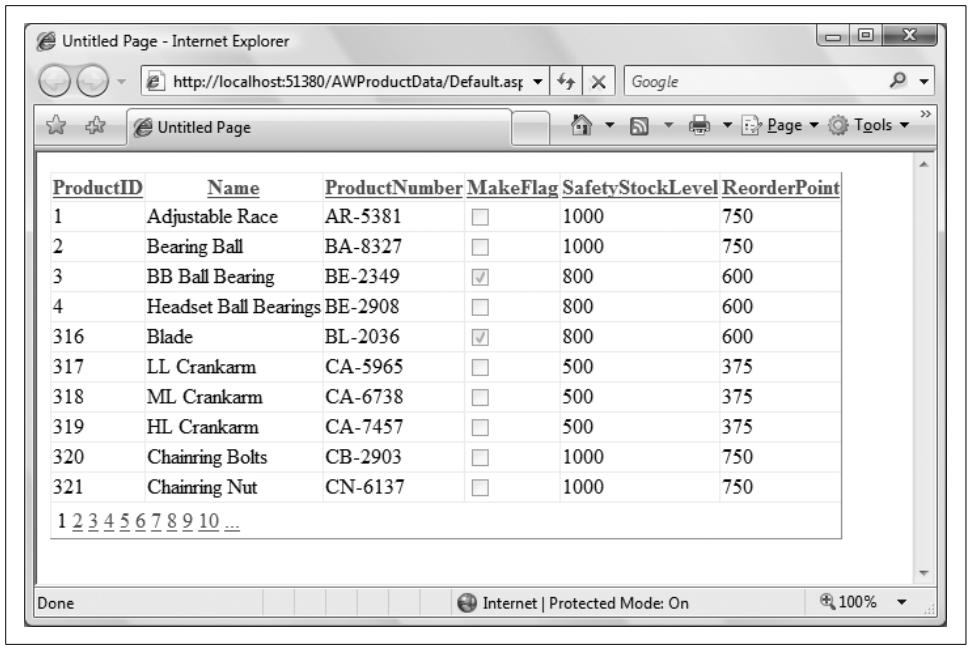
Once the data source is set, the data grid is redrawn, with a column for each field returned by the data source. The column headers are filled in for you based on the column names in the table that the data source represents.

You'll have an opportunity to make the grid look much prettier in just a short while.

Open the Smart Tag again and check "Enable Paging," which allows the grid to show a limited number of entries on each page and provide links to the other pages providing access to all the data. Also check "Enable Sorting," which allows the user to sort the grid by clicking on a column header.

Set the page to be the start page for the application (right-click the page in the Solution Explorer and select "Set As Start Page") and then run the application. Figure 4-13 demonstrates how the screen should appear.

Notice that the MakeFlag column (which is a Boolean value of some obscure use to the AdventureWorks business model) is shown as a checkbox. Also note that the column headers are shown as links. Click on one of them now—you see that the grid is sorted by that column. Also notice that at the bottom of the grid are links to page through more data, 10 rows at a time. Click on some of those, too, to see the various pages.



ProductID	Name	ProductNumber	MakeFlag	SafetyStockLevel	ReorderPoint
1	Adjustable Race	AR-5381	☐	1000	750
2	Bearing Ball	BA-8327	☐	1000	750
3	BB Ball Bearing	BE-2349	☑	800	600
4	Headset Ball Bearings	BE-2908	☐	800	600
316	Blade	BL-2036	☑	800	600
317	LL Crankarm	CA-5965	☐	500	375
318	ML Crankarm	CA-6738	☐	500	375
319	HL Crankarm	CA-7457	☐	500	375
320	Chainring Bolts	CB-2903	☐	1000	750
321	Chainring Nut	CN-6137	☐	1000	750

1 2 3 4 5 6 7 8 9 10 ...

Figure 4-13. *With the GridView in place and connected to the data source, you can see the data you asked for. Notice the clickable headings and the paging links.*

Each time you click on one of the columns or one of the page numbers, the entire page is posted back to the server, and you'll encounter a noticeable delay and flicker. You know how to fix that!

Close the browser and return to Design view. Drag an `UpdatePanel` control onto the page from the AJAX Extensions section of the Toolbox. Drag the `SqlDataSource` and `GridView` controls already on the page into the `UpdatePanel`.

Run the application again. Notice there are no visible postbacks when you page or sort, and consequently, no flicker.

Auto-Generated Code

Switch to Source view and look at the markup code that was generated for the `GridView`. It should appear as highlighted in Example 4-1.

Example 4-1. *The GridView auto-generates a lot of control source code for you; you don't have to write any of this*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
      Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

Example 4-1. *The GridView auto-generates a lot of control source code for you; you don't have to write any of this (continued)*

```
<head runat="server">
   <title>Untitled Page</title>
</head>
<body>
   <form id="form1" runat="server">
   <asp:ScriptManager ID="ScriptManager1" runat="server">
   </asp:ScriptManager>
   <div>
      <asp:UpdatePanel ID="UpdatePanel1" runat="server">
         <ContentTemplate>
            <asp:SqlDataSource ID="SqlDataSource1" runat="server"
               ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString %>"
               SelectCommand="SELECT ProductID, Name, ProductNumber, MakeFlag,
                   SafetyStockLevel, ReorderPoint
               FROM Production.Product"></asp:SqlDataSource>
            <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
               AllowSorting="True" AutoGenerateColumns="False"
               DataKeyNames="ProductID" DataSourceID="SqlDataSource1">
               <Columns>
                  <asp:BoundField DataField="ProductID" HeaderText="ProductID"
                     InsertVisible="False"
                     ReadOnly="True" SortExpression="ProductID" />
                  <asp:BoundField DataField="Name" HeaderText="Name"
                     SortExpression="Name" />
                  <asp:BoundField DataField="ProductNumber"
                     HeaderText="ProductNumber" SortExpression="ProductNumber" />
                  <asp:CheckBoxField DataField="MakeFlag"
                     HeaderText="MakeFlag" SortExpression="MakeFlag" />
                  <asp:BoundField DataField="SafetyStockLevel"
                     HeaderText="SafetyStockLevel" SortExpression="SafetyStockLevel" />
                  <asp:BoundField DataField="ReorderPoint"
                     HeaderText="ReorderPoint" SortExpression="ReorderPoint" />
               </Columns>
            </asp:GridView>
         </ContentTemplate>
      </asp:UpdatePanel>
   </div>
   </form>
</body>
</html>
```

The IDE has done a lot of work for you. It has examined the data source and created a `BoundField` for each column in the data. Further, it has set the `HeaderText` to the name of the column in the database, represented by the `DataField` attribute. It has set the `AllowPaging` and `AllowSorting` properties to `true`. In addition, it has also set the `SortExpression` property to the name of the field. Finally, you'll notice on the declaration of the `GridView` that it has set `AutoGenerateColumns` to `False`.

If you were creating the GridView by hand, and if you wanted to let the grid create all the columns directly from the retrieved data, you could simplify the code by setting AutoGenerateColumns to True. (If AutoGenerateColumns is set to True, and you also include explicitly bound columns, then you will display duplicate data.)

To see this at work, create a second GridView by dragging another GridView control from the Toolbox inside the UpdatePanel, below the first. In the Smart Tag, set the Data Source to the same source as that of the first, SqlDataSource1. Click on the "Enable Paging" and "Enable Sorting" checkboxes.

Now go to Source view. If necessary, delete the <columns> collection from the new grid, GridView2. Change AutoGenerateColumns to the default value: True. The declaration for this second GridView should look something like the following:

```
<asp:GridView ID="GridView2" runat="server"
        AllowPaging="True" AllowSorting="True"
        DataSourceID="SqlDataSource1" >
</asp:GridView>
```

Run the page. Both grids behave identically and are visually indistinguishable. So why does the IDE create the more complex version? When you turn off AutoGenerateColumns, the IDE gives you much greater control over the presentation of your data. For example, you can set the headings on the columns (such as changing ProductNumber to Product No.). You can change the order of the columns or remove columns you don't need, and you can add new columns with controls for manipulating the rows.

You can make these changes by manually coding the HTML in the Source view, or by switching to Design view and clicking the Smart Tag for the GridView and choosing Edit Columns. Do that now for GridView1, and you'll see the Fields dialog box, as shown in Figure 4-14.

This dialog box is divided into three main areas: the list of available fields, the list of selected fields (with buttons to remove fields or reorder the list), and the properties window on the right. When you click on a selected field (such as ProductID), you can set the way that field will be displayed in the data grid (such as changing the header to ID).

While you're examining what you can do with the GridView, let's make it look a little nicer. First, delete or comment out the second (simpler) grid (GridView2) you just created a few moments ago. Second, open the Smart Tag on the original grid. Click AutoFormat and choose one of the formatting options. Of course, you can format it by hand, but why work so hard for a simple example? We'll choose "Brown Sugar" because it shows up well in the printed book. Run the application. The output should appear as in Figure 4-15.

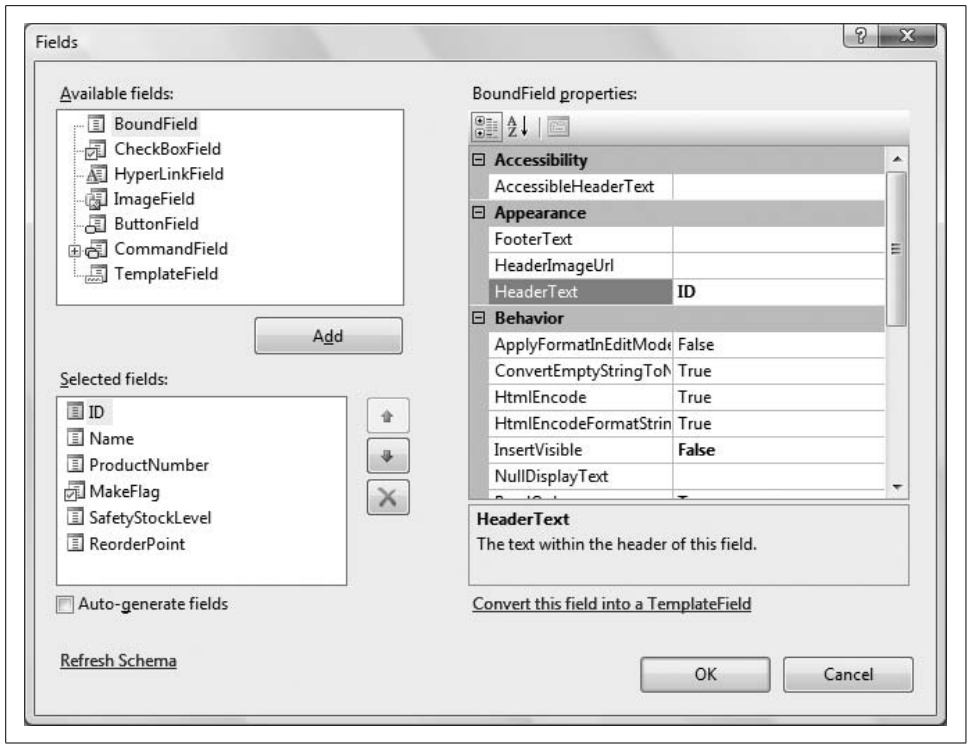


Figure 4-14. *The Fields dialog lets you change the properties of your data columns, without having to do it in Source view.*

Adding Insert, Update, and Delete Statements

At this point, the `SqlDataSource` you've created has only a `SELECT` statement to extract data from the database:

```
SelectCommand="SELECT ProductID, Name, ProductNumber,
   MakeFlag, SafetyStockLevel, ReorderPoint
   FROM Production.Product" >
```

That's fine, if all you want to do is display the data in the database. To take the next step, though, you probably want to be able to add new data, edit existing data, and even delete data.

You can do all that just as easily as you did the `SELECT` statement, by asking your data source control to generate the remaining Create, Retrieve, Update, and Delete statements (fondly known as CRUD statements), using a wizard to make your work easier.

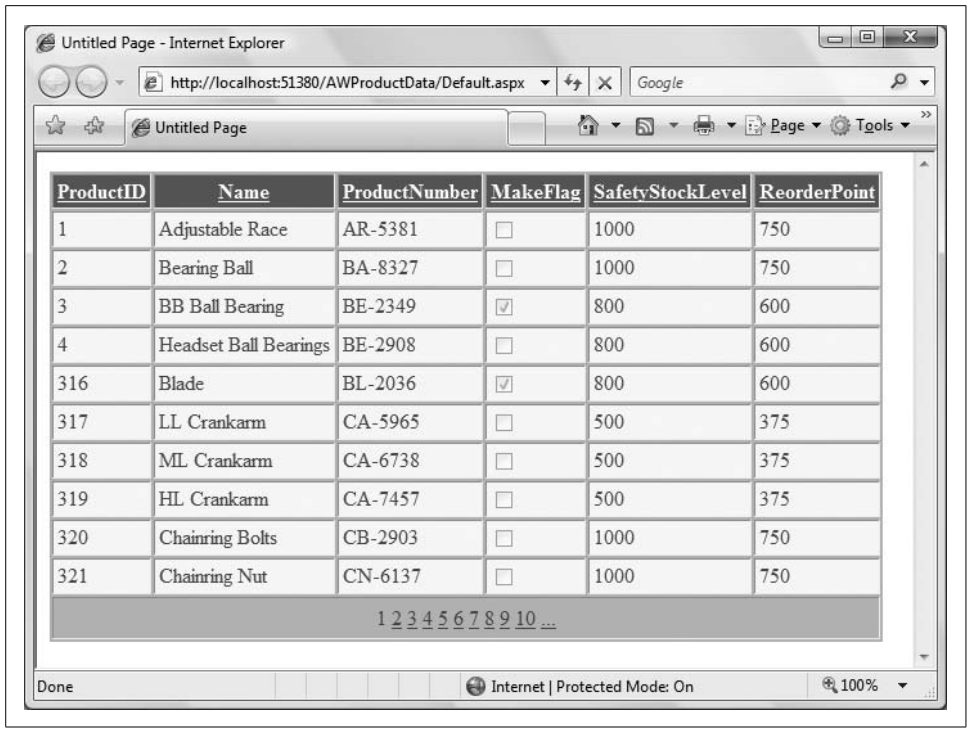


Figure 4-15. *The AutoFormat option in the GridView's Smart Tag lets you choose the formatting option that best fits your site, and applies it automatically.*

To see this in action, switch to Design view, click on the Smart Tag for the `SqlDataSource`, and choose Configure Data Source. The Configure Data Source Wizard opens, displaying your current connection string. Click Next; the Configure Select Statement dialog box is displayed, as shown earlier in Figure 4-8.

Recall the previous time you used the Configure Data Source Wizard—it did not correctly identify the table in the autogenerated `SELECT` statement, omitting the schema name. You worked around that by specifying your own SQL statement. Because the `SELECT` statement you needed was relatively simple to type in, that was not a problem.

However, there is a *lot* of typing involved for all the CRUD statements. So for the rest of these statements, you will use the Wizard to generate the SQL code, and then just fix the table names.

Make sure the "Specify columns from a table or view" radio button is selected, and the Product table is selected. Check the columns you want returned by the `SELECT` statement (`ProductID`, `Name`, `ProductNumber`, `MakeFlag`, `SafetyStockLevel`, `ReorderPoint`). This will create a new `SELECT` statement.

Click the Advanced button to open the Advanced SQL Generation Options dialog box. Select the "Generate INSERT, UPDATE, and DELETE statements" checkbox, as shown in Figure 4-16.

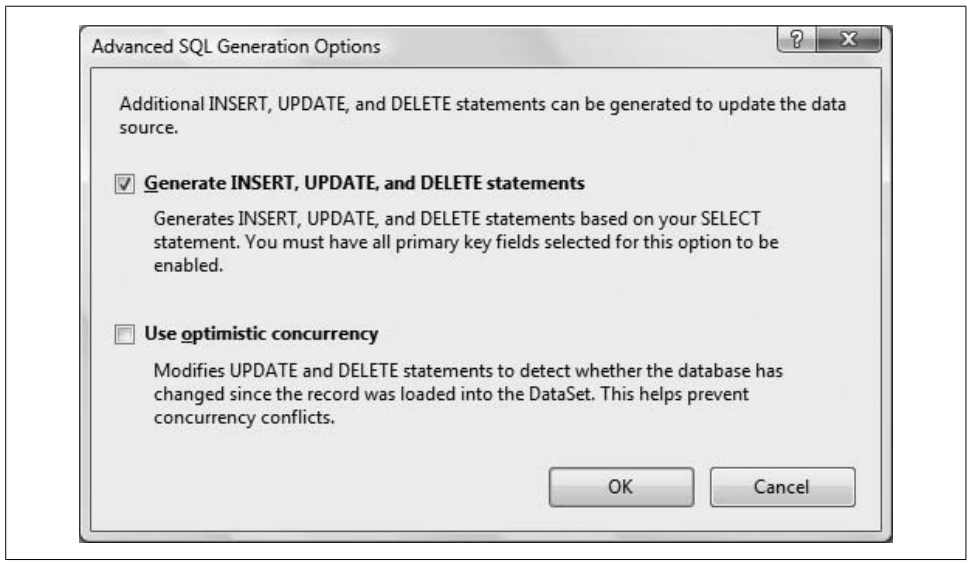

Advanced SQL Generation Options
Additional INSERT, UPDATE, and DELETE statements can be generated to update the data source.
Generate INSERT, UPDATE, and DELETE statements
Generates INSERT, UPDATE, and DELETE statements based on your SELECT statement. You must have all primary key fields selected for this option to be enabled.
Use optimistic concurrency
Modifies UPDATE and DELETE statements to detect whether the database has changed since the record was loaded into the DataSet. This helps prevent concurrency conflicts.
OK
Cancel

Figure 4-16. *You'll use the Advanced SQL Options dialog box to automatically create the SQL statements to add, edit, and delete data from your data source.*

Clicking this checkbox instructs the Wizard to create the remaining CRUD statements, and also enables the second checkbox, "Use optimistic concurrency." This is a feature that helps safeguard your data in case another user makes a change to the database at the same time you do. Leave this option unchecked for now and click OK. When you return to the Wizard, click Next, and then Finish. You may be asked to update your grid, which unfortunately will wipe out all your customization, but the good news is that you are now bound to a data source control that provides all four CRUD methods.

Open the Smart Tag on the `GridView` control again, and reapply the look and feel you want. Also—and this is important—select the checkboxes "Enable Editing" and "Enable Deleting."

Switch to Source view. The `SqlDataSource` markup will appear similar to Example 4-2, with the new SQL commands added. You'll still need to modify the table names, or else you'll get the error you saw earlier (see Figure 4-9). Add the schema name `[Production]` to each of the four statements highlighted in Example 4-2. (You can use Visual Studio's Quick Find feature (Ctrl + F) to quickly locate each instance of `[Product]` in your code.)

Example 4-2. *After you update the properties of the SqlDataSource control with CRUD statements, it should look like this*

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString %>"
    SelectCommand=" SELECT [ProductID], [Name], [ProductNumber],
                    [MakeFlag], [SafetyStockLevel], [ReorderPoint]
                    FROM [Production].[Product]"
    DeleteCommand="DELETE FROM [Production].[Product]
                    WHERE [ProductID] = @ProductID"
    InsertCommand="INSERT INTO [Production].[Product] ([Name],
                    [ProductNumber],
                    [MakeFlag], [SafetyStockLevel], [ReorderPoint])
                    VALUES (@Name, @ProductNumber, @MakeFlag,
                    @SafetyStockLevel,
                    @ReorderPoint)"
    UpdateCommand="UPDATE [Production].[Product] SET [Name] = @Name,
                    [ProductNumber] = @ProductNumber,
                    [MakeFlag] = @MakeFlag,
                    [SafetyStockLevel] = @SafetyStockLevel,
                    [ReorderPoint] = @ReorderPoint
                    WHERE [ProductID] = @ProductID" >
    <DeleteParameters>
        <asp:Parameter Name="ProductID" Type="Int32" />
    </DeleteParameters>
    <UpdateParameters>
        <asp:Parameter Name="Name" Type="String" />
        <asp:Parameter Name="ProductNumber" Type="String" />
        <asp:Parameter Name="MakeFlag" Type="Boolean" />
        <asp:Parameter Name="SafetyStockLevel" Type="Int16" />
        <asp:Parameter Name="ReorderPoint" Type="Int16" />
        <asp:Parameter Name="ProductID" Type="Int32" />
    </UpdateParameters>
    <InsertParameters>
        <asp:Parameter Name="Name" Type="String" />
        <asp:Parameter Name="ProductNumber" Type="String" />
        <asp:Parameter Name="MakeFlag" Type="Boolean" />
        <asp:Parameter Name="SafetyStockLevel" Type="Int16" />
        <asp:Parameter Name="ReorderPoint" Type="Int16" />
    </InsertParameters>
</asp:SqlDataSource>
```

Switch back to Design view and notice the Edit and Delete buttons on each row. They are the result of checking the Enable Editing and Enable Deleting checkboxes.

Taking apart the code in Example 4-2, on the first line is the declaration for the `SqlDataSource` (and its corresponding closing tag at the bottom). After the `ID`, the obligatory `runat="server"`, and the `ConnectionString` attribute, you see four attributes: the `SelectCommand` (which was there previously), and the new `DeleteCommand`, `InsertCommand`, and `UpdateCommand`.

If you run this web site and click on the Delete button to delete a row from the Product table, you will get the following error, raised by the database:

```
Sys.WebForms.PageRequestManagerServerErrorException:
The DELETE statement conflicted with the REFERENCE
constraint"FK_ProductInventory_Product_ProductID".
The conflict occurred in database "AdventureWorks",
table "Production.ProductInventory",
column 'ProductID'.
The statement has been terminated.
```

This error is caused by a *constraint* on the data, enforced by the database, which prevents the deletion of a record that is referenced elsewhere in the database. In other words, the product you are trying to delete is referenced in the `ProductInventory` table. If you were to delete that record from the `Product` table, there would be orphan records left behind in the `ProductInventory` table, which is a violation of the business rules of this database.

In a real-world application, the way to handle this would be to write some custom SQL code, called a stored procedure, which would first delete all the child records from the `ProductInventory` table, *then* delete the record from the `Product` table. The `SqlDataSource` control would then be configured to use that stored procedure for the Delete command, rather than the SQL statement it is currently configured to use. Unfortunately, stored procedures are beyond the scope of this book. However, our more advanced book, *Programming ASP.NET*, Fourth Edition, does cover this sort of situation.

The `DeleteCommand` takes a single parameter (`@ProductID`), which is specified in the `DeleteParameters` element:

```
<DeleteParameters>
    <asp:Parameter Name="ProductID" Type="Int32" />
</DeleteParameters>e
```

SQL CHEAT SHEET

Parameters

A *parameter* in a SQL statement allows parts of the statement to be replaced when the statement is actually run. SQL parameters are always preceded with the `@` symbol. So, in the following SQL statement:

```
delete from Products where ReorderPoint > @ReorderPoint
```

all of the records with a value of `ReorderPoint` greater than some specified value will be deleted from the Products table. One time the statement is run, that value might be 100, the next time it might be 5. The value depends on how the parameter was defined beforehand.

The `UpdateCommand` control requires more parameters, one for each column you'll be updating, as well as a parameter for `ProductID` (to make sure the correct record is updated). Similarly, the `InsertCommand` takes parameters for each column for the new record. All of these parameters are within the definition of the `SqlDataSource`.

Displaying and Updating the Data

Now that your `SqlDataSource` object is ready to go, you only have to set up your `GridView` control. In Design view, click on the `GridView` Smart Tag. Verify that the checkboxes to enable editing and deleting are selected, as shown in Figure 4-17.

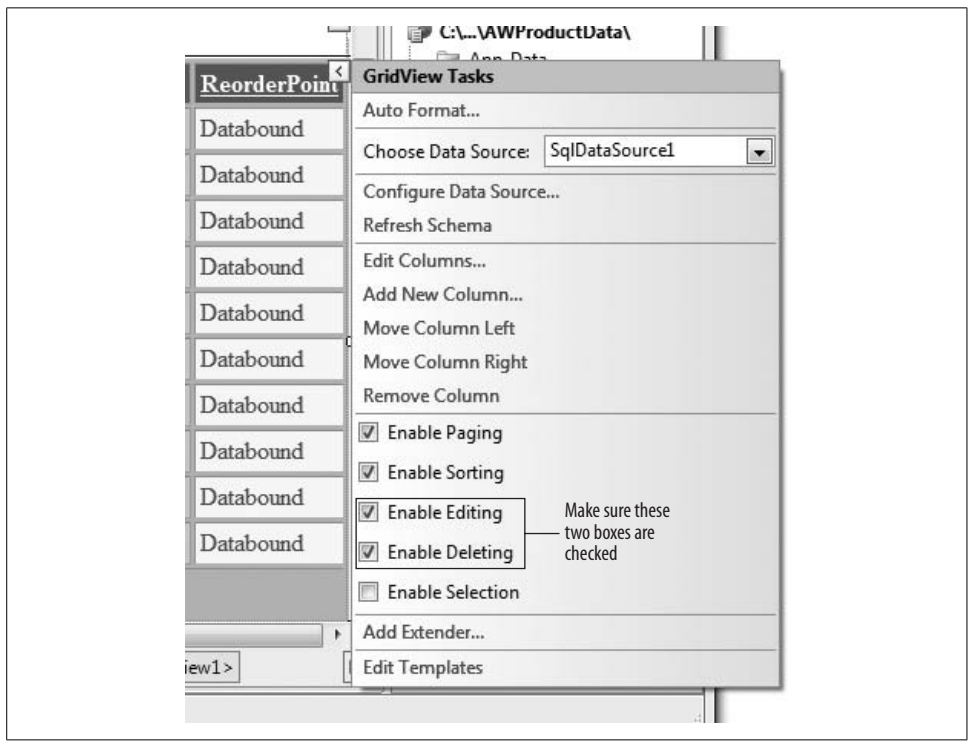


Figure 4-17. *Select the Smart Tag on the GridView, and check the boxes to enable editing and deleting.*

If you prefer to have buttons for Edit and Delete, rather than links, click on the Smart Tag and select "Edit Columns…." When the Fields dialog box opens, click the Command Field entry in the Selected Fields area (on the lower left). This brings up the Command Field Properties in the right-hand window. In the Appearance section of the Fields editor, choose `ButtonType` and then change Link to Button in the drop-down menu next to `ButtonType`, as shown in Figure 4-18.

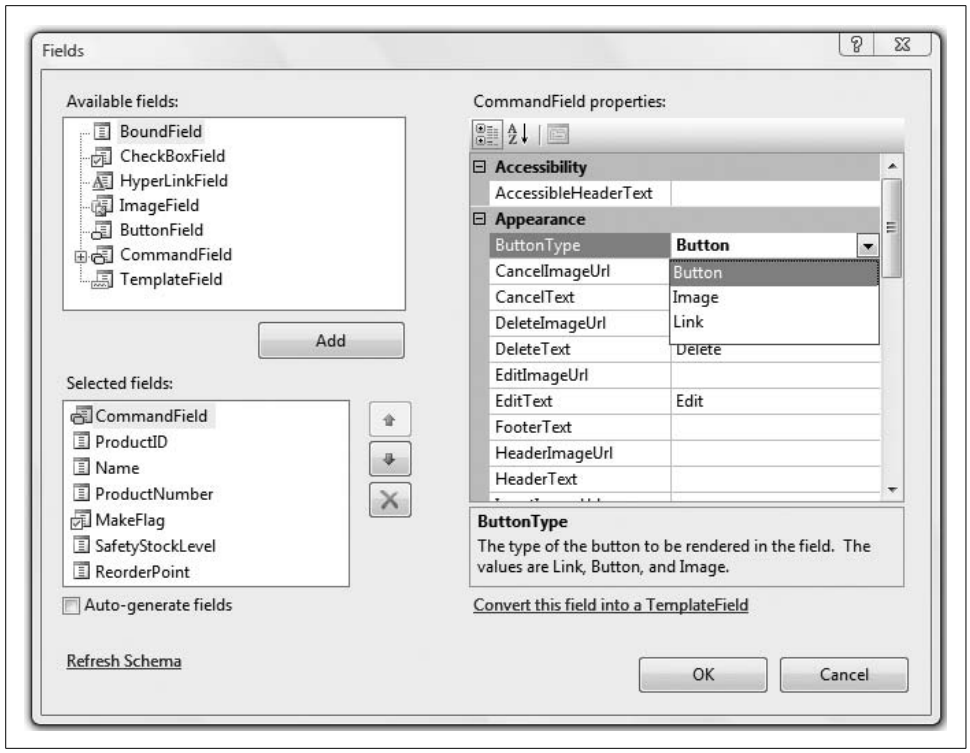


Figure 4-18. *Click the Smart Tag of the GridView, then click Edit Columns to get this Fields dialog box, where you can select and edit the columns in the GridView. Here, the CommandField button type is being changed.*

The result is that the commands (Edit and Delete) are shown as buttons, as shown in Figure 4-19.

Take It for a Spin

Start the application. The product database information is loaded into your `GridView`. When you click the Edit button, the data grid automatically enters edit mode. You'll notice that the text fields you can edit change to text boxes and checkboxes, as appropriate, and the command buttons change from Edit and Delete to Update and Cancel. Make a small change to one field, as shown in Figure 4-20.

When you click the Update button for that row, the grid and the database are both updated, which you can confirm by opening the table in the database, as shown in Figure 4-21.

To open the database table, stop the application first. Then, on the right side of the IDE, click the Database Explorer tab (it is called Server Explorer in VS). Expand the AdventureWorks node, and then expand the Tables node. Scroll down until you find

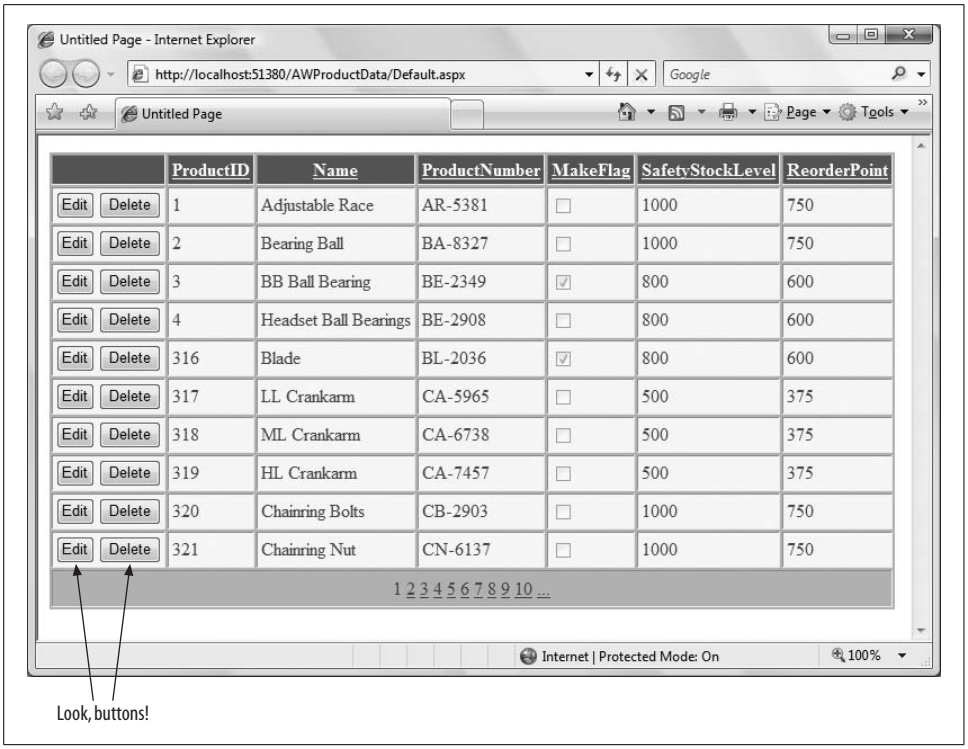

Untitled Page - Internet Explorer
http://localhost:51380/AWProductData/Default.aspx
Google
Untitled Page
Page
Tools
ProductID
Name
ProductNumber
MakeFlag
SafetyStockLevel
ReorderPoint
Edit Delete 1 Adjustable Race AR-5381 1000 750
Edit Delete 2 Bearing Ball BA-8327 1000 750
Edit Delete 3 BB Ball Bearing BE-2349 800 600
Edit Delete 4 Headset Ball Bearings BE-2908 800 600
Edit Delete 316 Blade BL-2036 800 600
Edit Delete 317 LL Crankarm CA-5965 500 375
Edit Delete 318 ML Crankarm CA-6738 500 375
Edit Delete 319 HL Crankarm CA-7457 500 375
Edit Delete 320 Chainring Bolts CB-2903 1000 750
Edit Delete 321 Chainring Nut CN-6137 1000 750
1 2 3 4 5 6 7 8 9 10 ...
Internet | Protected Mode: On
100%
Look, buttons!

Figure 4-19. *You can change the Edit and Delete links in the GridView to buttons, if you prefer.*

the `Product (Production)` table (in the IDE, the schema name is displayed in parentheses *after* the table name—go figure), then right-click it, and select "Show Table Data." This will show you the contents of the table from within the IDE.

Modifying the Grid Based on Conditions

Suppose you would like to modify the grid so the contents of the `Name` column are red when the `MakeFlag` column is checked, that is, when its value is `True`. In addition, you want all the ProductNumbers that begin with the letters CA to display in green. You can do this by handling the `RowDataBound` event. As the `GridView` is populated with data, each row of data is *bound* to the `GridView` individually, and the `RowDataBound` event is fired once for each row.

To modify the `GridView`, switch to Design view, click the `GridView`, click the lightning bolt in the Properties window, and double-click in the method name column (currently blank) to the right of the `RowDataBound` event. The IDE will create an event handler named `GridView1_RowDataBound( )` and then place you in the code-behind file within the skeleton of that method, ready for you to start typing code.

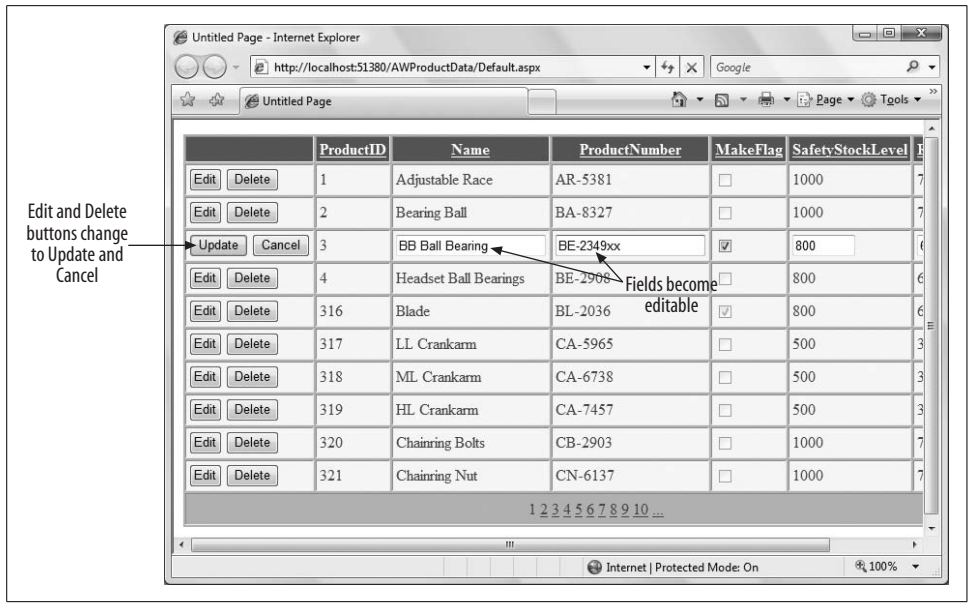


Figure 4-20. *When you click Edit on a row, that row enters edit mode. Any fields that can be edited change to text boxes and checkboxes.*

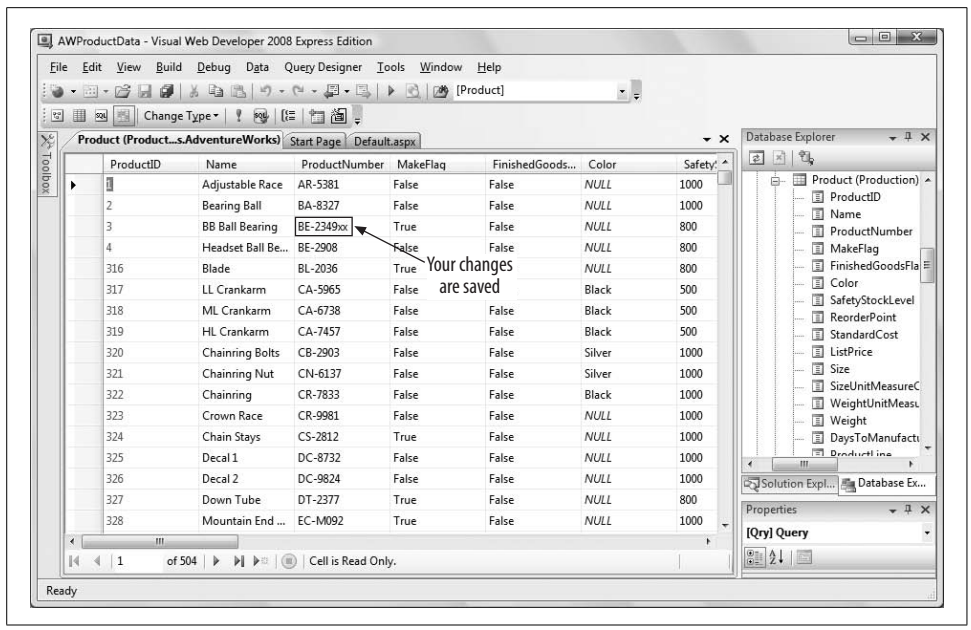

AWProductData - Visual Web Developer 2008 Express Edition
File Edit View Build Debug Data Query Designer Tools Window Help
[Product]
Change Type
Product (Product...s.AdventureWorks)
Start Page
Default.aspx
Toolbox
Database Explorer
ProductID Name ProductNumber MakeFlag FinishedGoods... Color Safety
1 Adjustable Race AR-5381 False False NULL 1000
2 Bearing Ball BA-8327 False False NULL 1000
3 BB Ball Bearing BE-2349xx True False NULL 800
4 Headset Ball Be... BE-2908 False False NULL 800
316 Blade BL-2036 True False NULL 800
317 LL Crankarm CA-5965 False False Black 500
318 ML Crankarm CA-6738 False False Black 500
319 HL Crankarm CA-7457 False False Black 500
320 Chainring Bolts CB-2903 False False Silver 1000
321 Chainring Nut CN-6137 False False Silver 1000
322 Chainring CR-7833 False False Black 1000
323 Crown Race CR-9981 False False NULL 1000
324 Chain Stays CS-2812 True False NULL 1000
325 Decal 1 DC-8732 False False NULL 1000
326 Decal 2 DC-9824 False False NULL 1000
327 Down Tube DT-2377 True False NULL 800
328 Mountain End ... EC-M092 True False NULL 1000
Your changes are saved
1 of 504
Cell is Read Only.
Product (Production)
ProductID
Name
ProductNumber
MakeFlag
Color
SafetyStockLevel
ReorderPoint
StandardCost
ListPrice
Size
Weight
Solution Expl...
Database Ex...
Properties
[Qry] Query
Ready

Figure 4-21. *If you view the table in the database after editing it in the GridView, you'll see that the changes have been saved.*

The second argument to this method is of type GridViewRowEventArgs. This object has useful information about the row that is being data bound, which is accessible through the Row property of the event argument.

Enter the code shown in Example 4-3.

Example 4-3. *The handler for the RowDataBound event lets you extract individual cells and work with them*

```
Protected Sub GridView1_RowDataBound(ByVal sender As Object, _
              ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs) _
              Handles GridView1.RowDataBound
   If e.Row.RowType = DataControlRowType.DataRow Then
      Dim cellProductNumber As TableCell = e.Row.Cells(3) ' ProductNumber column
      If cellProductNumber.Text.Substring(0, 2) = "CA" Then
          cellProductNumber.ForeColor = Drawing.Color.Green
      End If

      Dim cellMakeFlag As TableCell = e.Row.Cells(4) ' MakeFlag column
      Dim cb As CheckBox = CType(cellMakeFlag.Controls(0), CheckBox)
      If cb.Checked Then
          e.Row.Cells(2).ForeColor = Drawing.Color.Red
      End If
   End If
End Sub
```

The first If statement (highlighted in Example 4-3) tests if the type of Row passed in as a parameter—in other words, the row that was bound and triggered this event—is a DataRow (rather than a header, footer, or something else).

Once you know you are dealing with a DataRow, you can extract the cell(s) you want to examine from that row. Here, we will look at two cells: the ProductNumber cell is the fourth cell in the row, at offset (index) 3, and the MakeFlag cell is the fifth cell in, at offset 4. (Remember, all indexes are zero-based.)

To access the ProductNumber cell, you define a new variable, cellProductNumber, defined as a TableCell with the As keyword, and set it equal to the cell at offset 3 in the row, like this:

```
Dim cellProductNumber As TableCell = e.Row.Cells(3)
```

Once you have the cell as a variable, you want to get the text contained in the cell to compare to your known value. You do that by accessing the Text property of cellProductNumber, and then using the Substring( ) function.

The Substring( ) function, as you might guess from its name, extracts a smaller string from a larger one. This is a pretty simple function to work with. First, you call the function on a string, and you give it two numbers as parameters: the index of the start of the substring, and the length of the substring. As with all other indexes, the first character in the string is position zero. You want the first two characters from the Text string, so the starting index is 0, and the length of the substring is 2. Therefore, to get

the first two characters from your string, you use the function `Substring(0,2)`. Once you have that substring, you can use a simple `If` statement to compare it to the string you want to match, "CA":

```
If cellProductNumber.Text.Substring(0, 2) = "CA" Then
```

It there is a match, you want to set the `ForeColor` property of the cell to green, which you can do using the `Drawing.Color.Green` property:

```
cellProductNumber.ForeColor = Drawing.Color.Green
```

In the case of the MakeFlag field, it is somewhat more complicated. It's easy enough to isolate the cell that contains the checkbox—it's at index 4—and then assign that value to a new variable called `cellMakeFlag`:

```
Dim cellMakeFlag As TableCell = e.Row.Cells(4)
```

This is the same technique you used to isolate the ProductNumber cell. In this case, though, the `Text` property of this cell will always be empty. However, the cell does contain a `CheckBox` control, which is the only control in the cell. Instead of reading the *text* in the cell, you want to read the *value* of the `Checked` property of that `CheckBox` control. Each cell has a collection of all the controls contained in the cell, called `Controls`, which has a zero-based index. Because the checkbox you want is the only control in the collection, you know it's at `cellMakeFlag.Controls(0)`. Next, you define a new variable, `cb`, which you define as a `CheckBox`. Then, you use the `CType` function on the control you just isolated, to convert the control to a `CheckBox`. This works because we know it is a `CheckBox`:

```
Dim cb As CheckBox = CType(cellMakeFlag.Controls(0), CheckBox)
```

Then, you test the `Checked` property of the `CheckBox`:

```
If cb.Checked Then
```

If the box is checked, `cb.Checked` will evaluate to `true`. If it is checked, you want to set the `ForeColor` property of the third cell in the row (offset 2), the `ProductName` column:

```
e.Row.Cells(2).ForeColor = Drawing.Color.Red
```

You set the color of the cell the same way you did for `ProductNumber`, but notice this time you're not changing the color of the checkbox cell itself—you're changing a different cell in the table.

Run the web site. It will look identical to Figure 4-19 (shown earlier in this chapter), except the product names for which the `MakeFlag` field is checked will display in red, and some of the product numbers will display in green, although you may need to page through the data to find them. (Neither of these changes will be obvious in the printed book, so we will forego a figure showing the color changes.)

VB CHEAT SHEET

CType Method

CType converts its first argument into an object of a new type as specified by its second argument. In the case shown here, it is converting an object to a CheckBox. If the object you pass is not of the appropriate type, CType generates an error. Read this statement:

```
Dim cb As CheckBox = CType(cellMakeFlag.Controls(0), CheckBox)
```

as follows: "Find the first item in the Controls collection in cellMakeFlag and convert it to type CheckBox." The result will be an object of type CheckBox or an exception will be thrown. If no exception is thrown, assign the result to the variable cb, which is of type CheckBox.

If you want to be extra careful, you can wrap the CType conversion in a Try/Catch block, discussed in Chapter 8, but that isn't really necessary here as you know it is a checkbox.

Note to C# users: This is one of the few methods available in VB but not in C#. Instead, in C#, you *cast* from one type to another by enclosing the target type in parentheses, as in the following C# statement equivalent to the one above in VB:

```
CheckBox cb = (CheckBox)cellMakeFlag.Controls[0];
```

Selecting Data from the GridView

Often you need to select a row from the grid and extract data from that row. This is easy to do using the SelectedIndexChanged event of the GridView.

To see how this works, drag a Label control from the Standard section of the Toolbox onto the Design view, below the grid, but within the UpdatePanel control. Change the Text property of this Label to Name. Then, drag a TextBox control next to the Label. Change its ID property to txtName and set its ReadOnly property to True. You now have a place to display the name of the selected item from the grid.

Click on the Smart Tag of the GridView and check the "Enable Selection" checkbox. This will cause a Select button to display in the first column of the grid, next to the Edit and Delete buttons already there, as shown in Figure 4-22.

Now all you need to do is set up the event handler to respond to the Select buttons. Double-click on the Select button in the first row of the grid. This will open up the code-behind file with the skeleton of the SelectedIndexChanged already created for you, ready to accept your custom code. Enter the highlighted code from the following snippet:

```
Protected Sub GridView1_SelectedIndexChanged(ByVal sender As Object, _
      ByVal e As System.EventArgs) _
      Handles GridView1.SelectedIndexChanged
   If GridView1.SelectedRow.RowType = DataControlRowType.DataRow Then
      Dim cellName As TableCell = GridView1.SelectedRow.Cells(2) ' Name column
```

```
            txtName.Text = cellName.Text
        End If
    End Sub
```

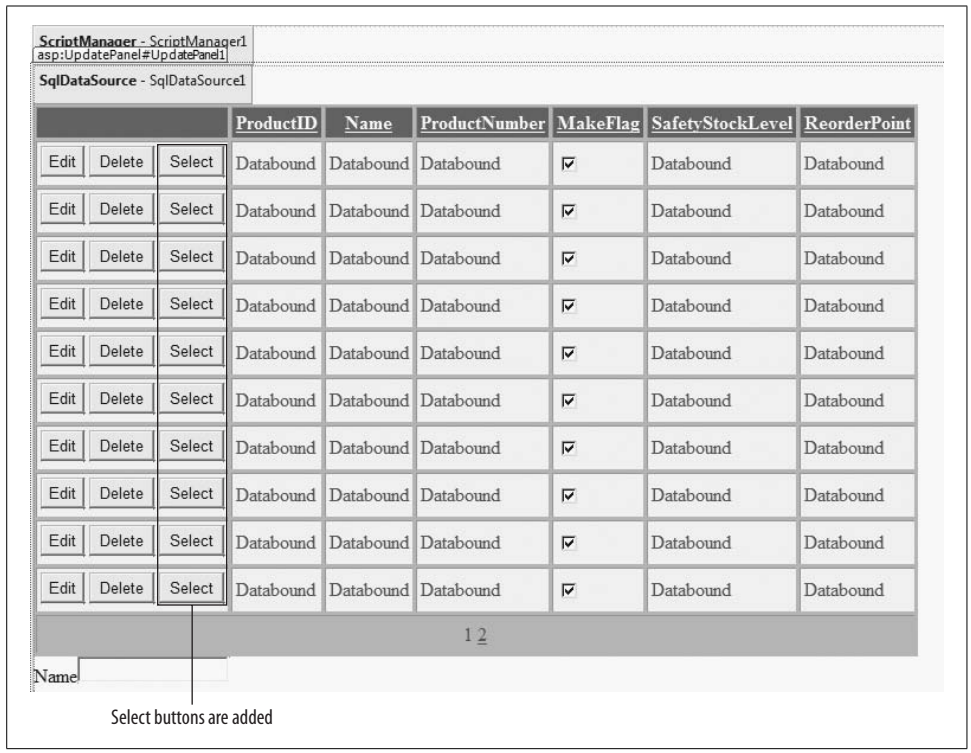


Figure 4-22. *Clicking Enable Selection in the Smart Tag causes Select buttons to appear in a GridView.*

This code first tests to determine if the selected row is a `DataRow` (as opposed to a `HeaderRow` or a `FooterRow`). If it is a `DataRow`, it creates a variable of type `TableCell`, to which you assign the third cell in the selected row (because of zero-based indexing, the third item will have an index value of 2). Then the `Text` property of the `TextBox` is set equal to the `Text` property of that cell.

Run the app and click on one of the Select buttons. The name from the selected row appears in the `TextBox`.

Passing Parameters to the SELECT Query

Sometimes you do not want to display all the records in a table. For example, you might want to have users select a product from your grid and display the order details for that product somewhere else on the current page. To do this, you'll need a way to select a product as well as a way to pass the `ID` of the selected product to the

second grid. The Select buttons are already in place from the previous example, so all you need to do now is pass the `ID` of the selected product to the second grid.

To keep your examples clear, copy the previous example, *AWProductData* to a new web site, *AWProductDataOrderDetails*.

See Appendix B for details about copying a web site.

You need to create a second `GridView`, which will be used to display the order details. From the Toolbox, drag the second `GridView` onto the page below the first, after the `Label` and `TextBox`, and inside the pre-existing `UpdatePanel`. Open the Smart Tag for the `GridView`. In the Choose Data Source field of the Smart Tag, select <New Data Source> this time instead of using an existing `DataSource` control. The Data Source Configuration Wizard appears, but with a new screen this time, shown in Figure 4-23. Select the Database option, and name the data source `AdventureWorksOrderDetails`.

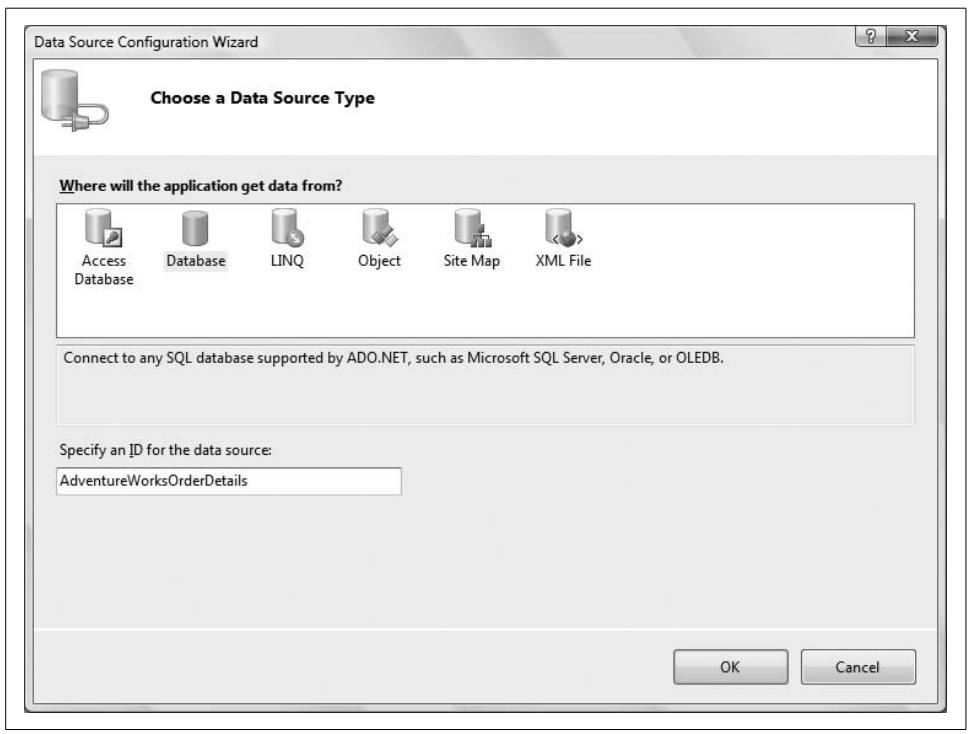


Figure 4-23. *You're creating a new data source control this time, instead of using one you've already placed on the page. Select Database as the type, and name it AdventureWorksOrderDetails.*

Click OK, and you'll see the connection string screen from Figure 4-4; use the existing connection string. Choose the `SalesOrderDetail` table, select the columns `SalesOrderID`, `CarrierTrackingNumber`, `OrderQty`, `UnitPrice`, `UnitPriceDiscount`, and `LineTotal`, and then click the WHERE button, as shown in Figure 4-24.

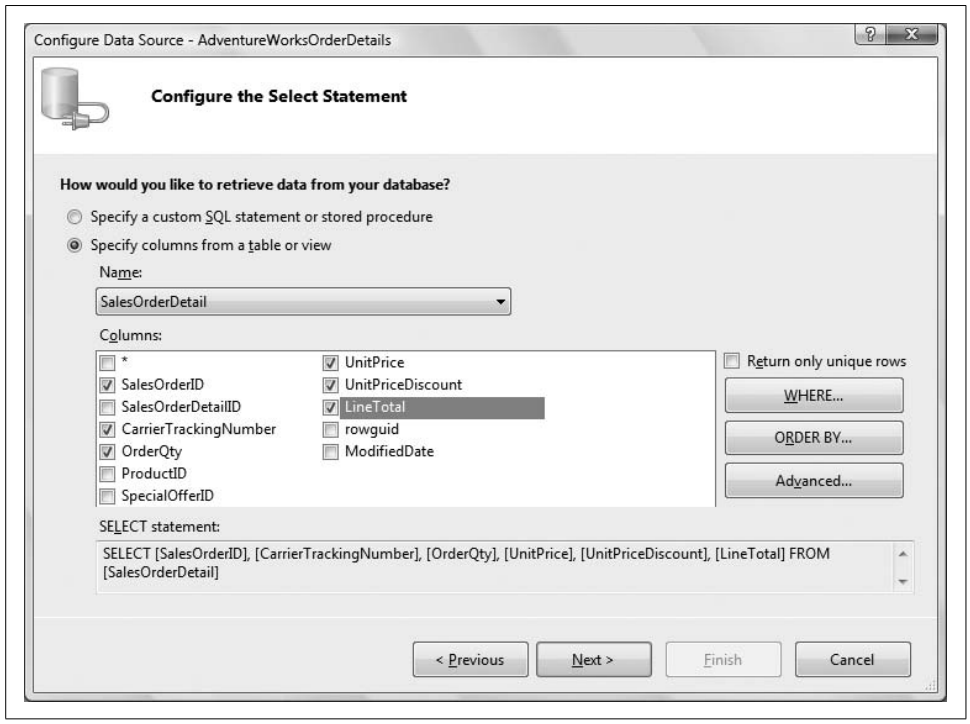


Figure 4-24. *Configuring the SalesOrderDetail table SELECT statement is similar to the way you set up the first DataSource, but this time, you'll add a WHERE clause.*

A `WHERE` clause is a SQL language construct used to narrow the set of data returned by the `SELECT` statement. In other words, you're saying, "Get me all the records from this table, *where* this condition is true." The condition could be defined any number of ways—where the amount in inventory is less than 10, where the customer name is "Smith," or where the copyright date is after 1985. It all depends on the types of information you have stored in your tables, and what you need for this circumstance.

When you click the WHERE button, the Add WHERE Clause dialog opens, which you can see in Figure 4-25. First, you pick the column you want to match on; in this case, `ProductID`. Next, pick the appropriate operator for your condition statement. Your choices include (among others): equal to, less than/greater than, like, and contains. For this exercise, use the default (=).

The third drop-down lets you pick the source for the `ProductID`—that is, where you will get the term you want to match on. You can pick from any one of several objects in the menu, or choose None if you'll be providing a source manually. In this case, you'll obtain the source of the `ProductID` from the first `GridView`, so choose `Control`.

When you choose `Control`, the Parameter properties panel of the dialog wakes up. You are asked to provide the `ID` of the `Control` containing the target parameter. Select `GridView1`. Once you've made all your choices, the screen will resemble Figure 4-25.

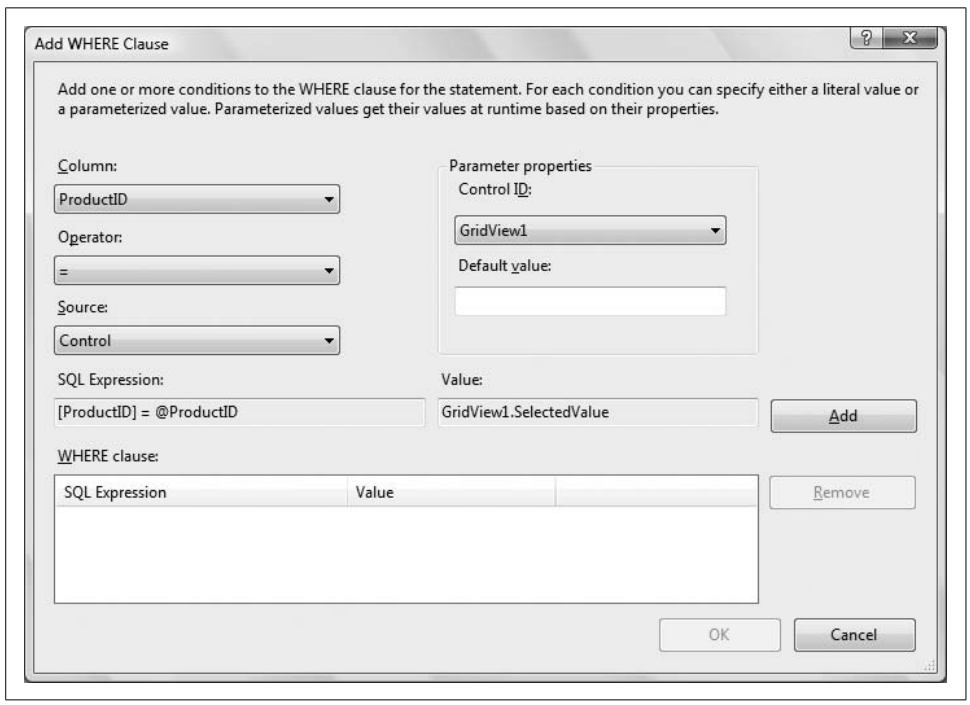


Figure 4-25. *Add a WHERE clause to your SELECT statement with the Add WHERE Clause dialog. You select the column, the operator, and the source here.*

Click Add. When you do, the upper portion of the dialog returns to its initial (blank) state, and the `WHERE` clause is added to the `WHERE` Clause window. You could add additional `WHERE` clauses at this point, to further restrict the data, but that is not necessary for this example.

Click OK to return to the ConfigureData Source dialog box. While you are at it, sort the results by the `SalesOrderID` column by clicking on the ORDER BY button. The Add ORDER BY Clause dialog with the `SalesOrderID` column selected is shown in Figure 4-26. The ORDER BY clause is another SQL construct, and this one does just what its name implies—it sorts the results using the specified field for the sort order.

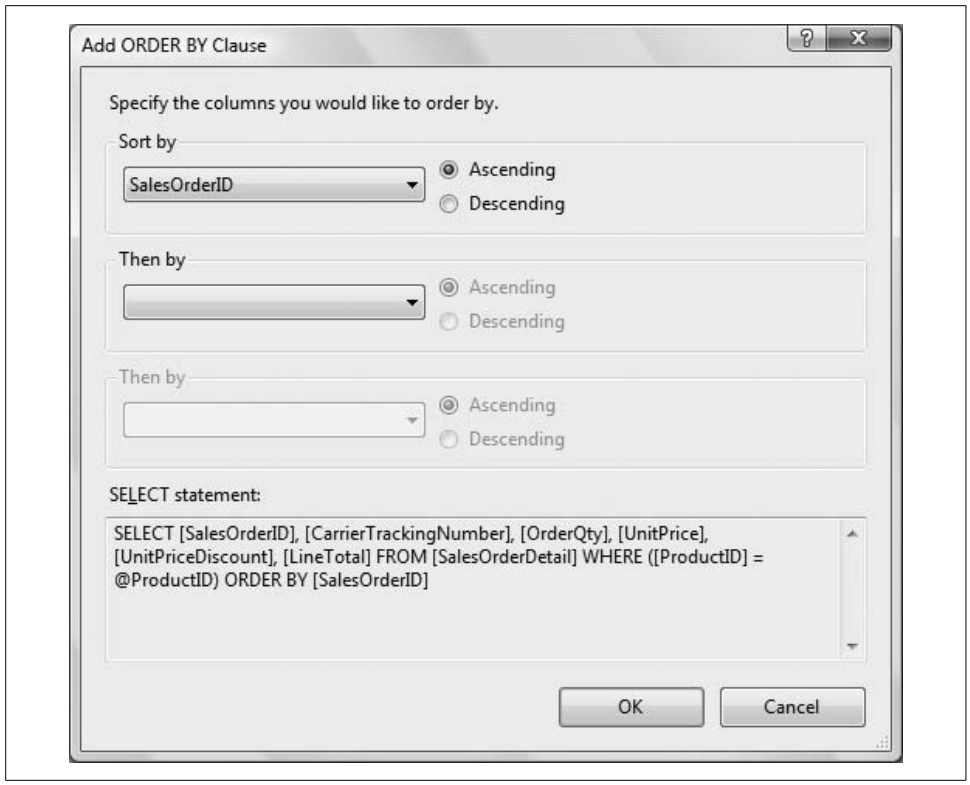


Figure 4-26. *Add an ORDER BY clause to sort the results of your SELECT statement.*

Click OK, then Next, then Finish, and the Configure Data Source Wizard is finished.

Switch to Source view and again fix the name of the tables in the SQL statements that were auto-generated. This time, you'll need to add `[Sales]` in front of the `[SalesOrderDetail]` item because this data is from the Sales table. The markup for the second `GridView` and its associated `SqlDataSource` is shown in Example 4-4, with the corrected table name highlighted. Also highlighted are the results of the WHERE and ORDER BY buttons from the Configure Data Source Wizard.

Example 4-4. *Visual Studio supplies the select parameters for you, but you still need to fix the schema in the second GridView*

```
<asp:GridView ID="GridView2" runat="server"
   DataSourceID="AdventureWorksOrderDetails">
</asp:GridView>
<asp:SqlDataSource ID="AdventureWorksOrderDetails" runat="server"
   ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString %>"
   SelectCommand="SELECT [SalesOrderID], [CarrierTrackingNumber], [OrderQty],
      [UnitPrice], [UnitPriceDiscount], [LineTotal]
      FROM [Sales].[SalesOrderDetail]
      WHERE ([ProductID] = @ProductID)
      ORDER BY [SalesOrderID]">
```

Example 4-4. *Visual Studio supplies the select parameters for you, but you still need to fix the schema in the second GridView (continued)*

```
    <SelectParameters>
        <asp:ControlParameter
            ControlID="GridView1"
            Name="ProductID"
            PropertyName="SelectedValue"
            Type="Int32" />
    </SelectParameters>
</asp:SqlDataSource>
```

The `SELECT` statement now has a `WHERE` clause that includes a parameterized value (`@ProductID`). In addition, within the definition of the `SqlDataSource` control is a definition of the `SelectParameters`. This includes one parameter of type `ControlParameter`, which is a parameter that knows how to get its value from a control (in our example, `GridView1`). In addition, a second property, `PropertyName`, tells it which property in the `GridView` to check. A third property, `Type`, tells it that the type of the value it is getting is of type `Int32`, so it can properly pass that parameter to the `SELECT` statement.

You may now reformat your grid and edit the columns as you did for the first grid, and then try out your new page, which should look something like Figure 4-27.

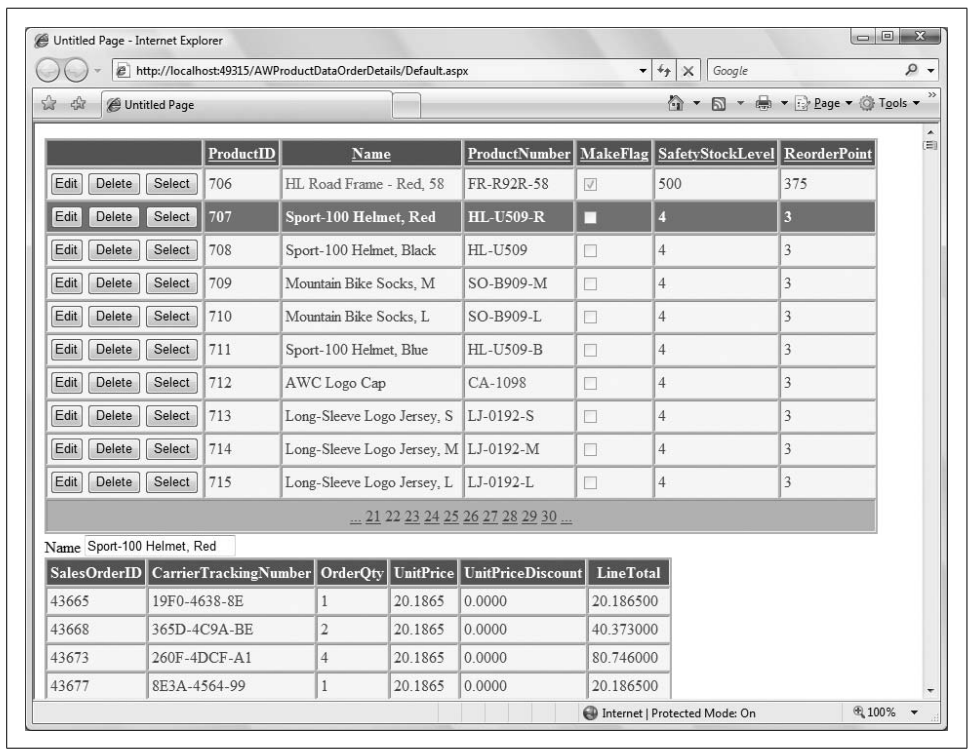

Untitled Page - Internet Explorer
http://localhost:49315/AWProductDataOrderDetails/Default.aspx
Google
Untitled Page
Page
Tools
ProductID
Name
ProductNumber
MakeFlag
SafetyStockLevel
ReorderPoint
Edit Delete Select 706 HL Road Frame - Red, 58 FR-R92R-58 500 375
Edit Delete Select 707 Sport-100 Helmet, Red HL-U509-R 4 3
Edit Delete Select 708 Sport-100 Helmet, Black HL-U509 4 3
Edit Delete Select 709 Mountain Bike Socks, M SO-B909-M 4 3
Edit Delete Select 710 Mountain Bike Socks, L SO-B909-L 4 3
Edit Delete Select 711 Sport-100 Helmet, Blue HL-U509-B 4 3
Edit Delete Select 712 AWC Logo Cap CA-1098 4 3
Edit Delete Select 713 Long-Sleeve Logo Jersey, S LJ-0192-S 4 3
Edit Delete Select 714 Long-Sleeve Logo Jersey, M LJ-0192-M 4 3
Edit Delete Select 715 Long-Sleeve Logo Jersey, L LJ-0192-L 4 3
... 21 22 23 24 25 26 27 28 29 30 ...
Name Sport-100 Helmet, Red
SalesOrderID CarrierTrackingNumber OrderQty UnitPrice UnitPriceDiscount LineTotal
43665 19F0-4638-8E 1 20.1865 0.0000 20.186500
43668 365D-4C9A-BE 2 20.1865 0.0000 40.373000
43673 260F-4DCF-A1 4 20.1865 0.0000 80.746000
43677 8E3A-4564-99 1 20.1865 0.0000 20.186500
Internet | Protected Mode: On
100%

Figure 4-27. *When you select a product in the first grid, the order details appear below in the second grid.*

The AdventureWorks database has no order details for any of the entries with ProductIDs below 707. The first entry with details is on page 22 of the grid, so be sure to move to page 22 (or later) to see product details (or click the ProductID header to see the products in descending order). If you select a product that does not have any order details, the second grid will *not* appear.

LINQ

As we mentioned earlier, most relational databases use some variant of SQL to retrieve and manipulate data. Much of the business logic of the application is contained within this SQL code—either as SQL statements passed to the database by your web app for execution, or as calls to stored procedures on the database server. A *stored procedure* is a predefined set of instructions that you can call from somewhere else in your code. Because you can dynamically build the SQL statements and pass parameters at runtime, this is a very powerful and flexible paradigm.

However, it has its shortcomings. The first is the headache of maintaining the code. In almost any typically complex web site, you probably want to implement most, if not all, of the database logic in stored procedures. This means that, at best, you will have source code in two separate locations—the file system where all your markup, code behind, and associated files live, and the database itself, where the stored procedures are. Even assuming that you have a suitable system in place for source code management (which in itself is the topic of many books), before you can implement features or fix bugs, you (the developer) must first determine if that piece of logic is handled on the file system or in the database.

A second, subtle but more important shortcoming has to do with what is referred to as the *impedance mismatch* between the object-oriented nature of .NET projects versus the table-oriented nature of relational data. In other words, in an object-oriented environment such as .NET, you work with instances of classes called *objects*. These classes may have *properties*, which describe the object, and *methods*, which define actions of the object. On the other hand, databases are constructed of tables, which contain rows and columns.

For example, you might have a class called `Dog`, with the properties of `Breed`, `Color`, and `Weight`. It also might have methods to create a new `Dog`, retrieve or modify a preexisting `Dog`, and save a `Dog` to the database. So your code might create an instance of a `Dog`, called `coco`, set some properties, and save the `Dog` to the database with code similar to the following:

```
Dim coco As Dog = New Dog( )
coco.Breed = "Black Lab"
coco.Color = Drawing.Color.Black
coco.Weight = 52
coco.Save( )
```

The `Save()` method would have to handle the not-so-trivial chore of converting the object into data that the database can save. Typically, this means the object itself would be represented by a row in a table, and the properties would be represented by columns within that row. Likewise, when retrieving data from the database, that data must be converted to the appropriate objects and properties for your program to work with it.

To address these issues, the .NET Framework 3.5 introduces a new language capability called *LINQ*, which stands for Language Integrated Query. LINQ is now part of all .NET 3.5 languages, including VB.NET and C#. It integrates query capability directly into the language as a first class feature, just like .NET languages include features such as data types, events, and conditional logic. Using LINQ, you can now interact with your data in a truly object-oriented fashion, writing code in VB or C# rather than in SQL.

To accomplish LINQ, several new language features have been added, such as type inference, anonymous types, and lambda expressions, which we are not going to cover in this book. LINQ is a big topic—it merits its own books, as well as chapters in several other books, including *Programming .NET 3.5* by Jesse Liberty and Alex Horovitz, *Programming C# 3.0,* Fifth Edition, by Jesse Liberty and Donald Xie, and *Programming Visual Basic 2008* by Tim Patrick, all published by O'Reilly.

To go along with LINQ, VS 2008 and VWD 2008 provide some significant new tools, including the Object Relational Designer and the `LinqDataSource` control.

LINQ is an advanced topic for developers, and the classes it employs are well beyond anything we use elsewhere in this book. In that sense, our discussion here goes against the "minimal coding" promise that we made at the beginning of this book. There's no other way to discuss LINQ except in terms of the classes it creates, however. In addition, we feel that LINQ is an interesting technology that addresses a couple of problems, but it's not yet developed enough to be useful at the beginner level. We're including it here because it's a major part of ASP.NET 3.5 that you should know about, but we don't think you're likely to use it much in its current version.

Creating the Object Model

LINQ works with an *object model*, a set of classes mapped to your database. You can create those classes manually by typing in a lot of code (IntelliSense will help), or you can let the IDE do most of the heavy lifting by using the Object Relational Designer. The Object Relational Designer, or O/R Designer, is a visual tool that performs two crucial tasks.

First, it creates the classes, or *entities*, that represent a database. Each table is mapped to its own class. Within that class, there will be property for each column in the table.

The properties are *strongly typed*. In other words, the data type of each property matches the data type of the corresponding column in the database table. This is important because it provides one of the big benefits of working with LINQ. Suppose the table has a column called `ProductID`, which is an integer (that would be an `int` in Transact SQL). The corresponding property in VB would of type `Integer`. This would be reflected by IntelliSense anywhere it was appropriate to use this property, and if you attempted to assign, say, a string to this property, the IDE would immediately flag this as invalid code and it would not build.

The other important task performed by the O/R Designer is to create a `DataContext`, which is an object used to send and receive data between your entities and the database. In other words, the `DataContext` is the bridge between the LINQ object model and the actual database. This will all become clear in a moment when we show you an example.

Just as ASP.NET provides a `SqlDataSource` control to provide easy access to data using SQL commands, it also provides a `LinqDataSource` control to provide similar access using LINQ. Once you have created your object model and `DataContext` with the O/R Designer, the `LinqDataSource` control can use that to be a binding source to other controls, such as a `GridView`.

To see this in action, create a new web site. Call it *LinqDataSource*. Select the Website → Add new Item… menu command. When the Add New Item dialog box comes up, select the LINQ to SQL Classes template. The default name will be *DataClasses.dbml*. Change this to *AdventureWorks.dbml*, and click the Add button. You will immediately be presented with the warning dialog box shown in Figure 4-28. Click Yes to accept this recommendation. (We will explain this in a moment.)

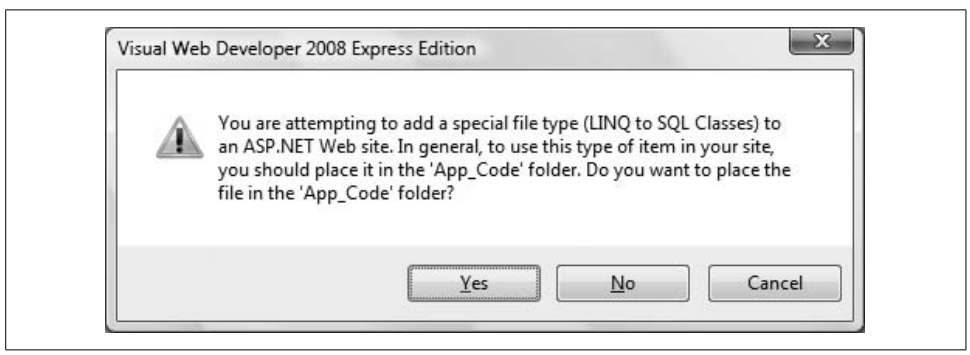

Visual Web Developer 2008 Express Edition
You are attempting to add a special file type (LINQ to SQL Classes) to an ASP.NET Web site. In general, to use this type of item in your site, you should place it in the 'App_Code' folder. Do you want to place the file in the 'App_Code' folder?
Yes
No
Cancel

Figure 4-28. *When you create a LINQ to SQL Classes file, you are prompted to place it in a special folder. Click Yes to accept this recommendation.*

After the IDE cooks for a bit, you will see in the Solution Explorer a new folder called *App_Code* as part of the web site. Within this folder will be a file called *AdventureWorks.dbml* with a plus sign next to it. Clicking on that plus sign expands it out to show two associated files also in the *App_Code* folder, *AdventureWorks.dbml.layout* and *AdventureWorks.designer.vb*, as shown in Figure 4-29.

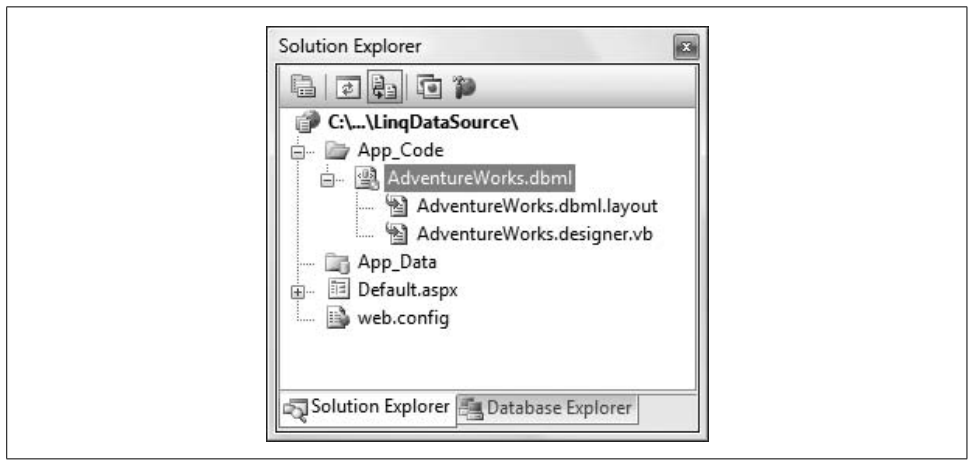


Figure 4-29. *After you add a LINQ to SQL Classes file, three new files will appear in an App_Code fold.*

The design surface of the IDE will now be showing the Object Relational Designer, which contains two panes side by side. Clicking anywhere within the O/R Designer will show the properties of the `AdventureWorksDataContext` (whew! that's a mouthful) in the Properties window. The IDE will now look something like that shown in Figure 4-30.

A lot has just happened here, so let's look at this a bit more carefully.

First, when you added the new item to the web site, the name of the template was "LINQ To SQL Classes". Notice it is plural, which might seem to be a typo. Remember, however, that the O/R Designer maps each individual database table to its own class. So, in the typical case, there will be multiple classes contained in this single item. Those classes will all be created for you in the file *AdventureWorks.designer.vb*.

Open this file in the IDE by double-clicking it in the Solution Explorer. A lengthy comment at the top reminds you that this file is auto-generated and warns against making any changes. At this point, the file contains just a small amount of boilerplate code, as you have not yet added any tables to the O/R Designer. The key point to notice here is that this source code declares a class called `AdventureWorksDataContext`, which inherits from `System.Data.Linq.DataContext`:

```
Partial Public Class AdventureWorksDataContext
    Inherits System.Data.Linq.DataContext
```

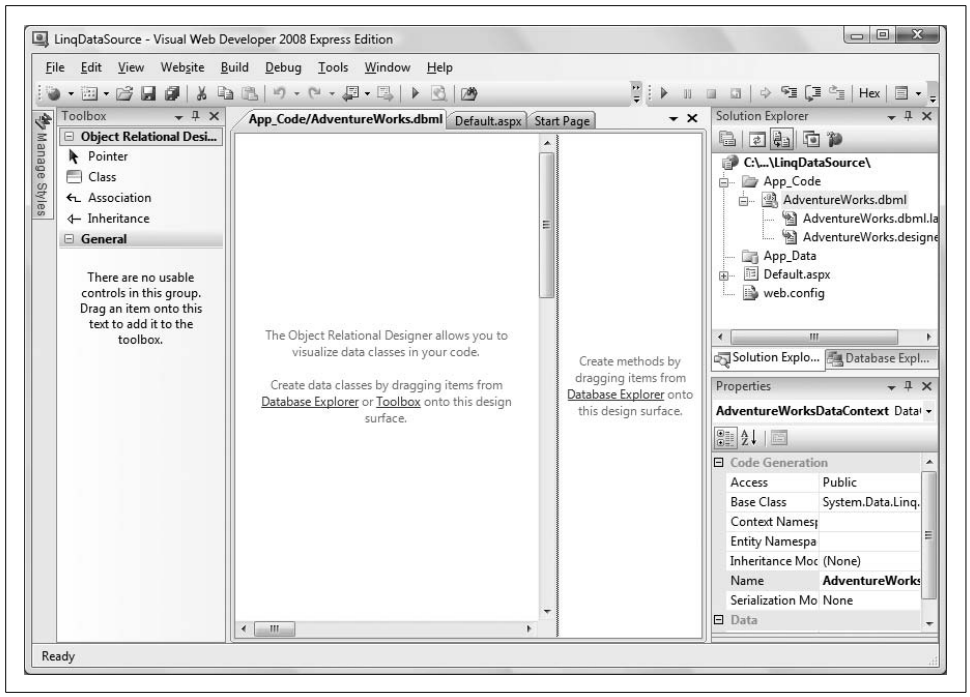


Figure 4-30. *After creating the LINQ to SQL Classes files, the O/R Designer will open in the IDE.*

Remember, the `DataContext` is the bridge between the object model, or classes, and the database. The class name is created automatically from the name you gave to the "LINQ to SQL Classes" item you added to the web site, concatenated with `DataContext`. Close this file when you've finished looking at it.

In Figure 4-30, the *AdventureWorks.dbml* file is selected in the Solution Explorer, which displays the O/R Designer on the design surface. The Properties window shows the properties of this object model. This reflects details of the class declaration in the above code snippet.

The next question you might ask is "Why put these files into the special folder called *App_Code*?". For any class to be available to a .NET application, the class must be compiled into a *dll*, or dynamic link library, and properly referenced by the application. Fortunately, the .NET Framework and VS/VWD makes it easy to accomplish this by accommodating two special folder names within the project. The first special folder name is *bin*. Any pre-compiled *dll* files contained within the *bin* folder are automatically referenced by the application. The second special folder is the aforementioned *App_Code* folder. Any class source code files, typically with an extension of *.vb* (for an app written with VB.NET; the equivalent extension for a C# app is *.cs*) in the *App_Code* folder will automatically be compiled and referenced when the application is built.

Along the left side of Figure 4-30, the Toolbox contains controls available to use in the O/R Designer. You can create your own classes by dragging the `Class` control onto the design surface. You can add properties to that class by right-clicking the class and selecting Add → Property. You can also create relationships between classes with the `Association` and `Inheritance` controls. We will not be using these controls in this book, although they can be very useful in more advanced scenarios.

With the O/R Designer open, you are ready to start building your object model from the database. Open the Database Explorer (Server Explorer in Visual Studio). You should already have a Data Connection to the AdventureWorks database from the previous examples in this chapter. If not, right-click on Data Connections and click Add Connection…. Follow the steps outlined in Figure 4-5 to create the connection.

Expand the connection to the AdventureWorks database by clicking on the plus sign next to it. Then, expand the list of tables by clicking the plus sign next to Tables. Scroll down to find the Product table and drag that table onto the left pane of the O/R Designer. The screen should like something like Figure 4-31.

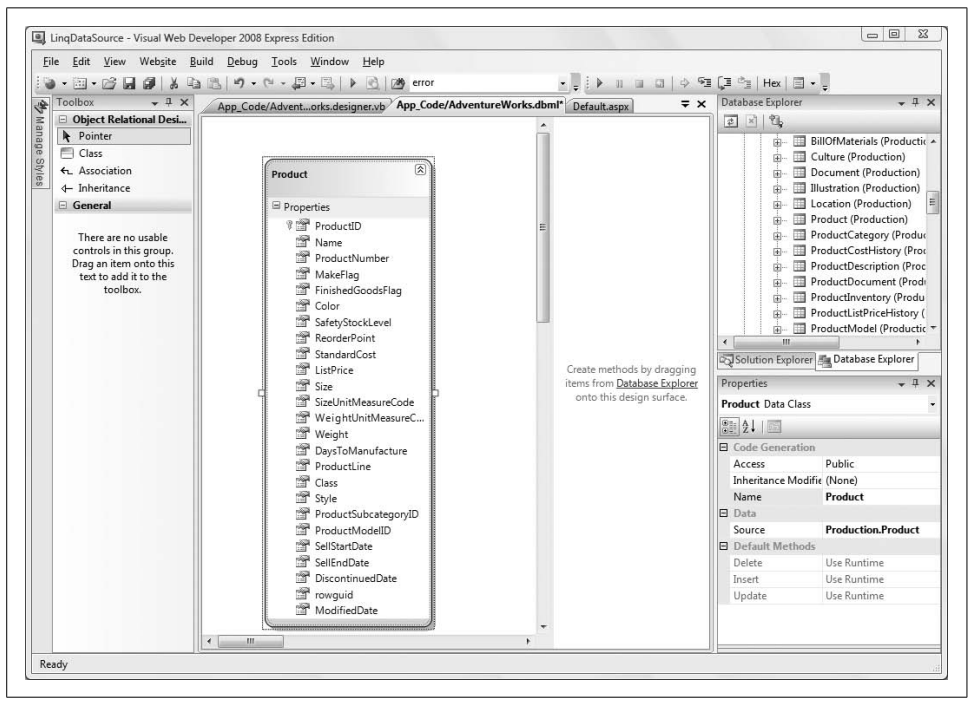


Figure 4-31. *Drag a table from the Database Explorer onto the O/R Designer to create a class representing that table.*

While you're at it, drag a few more tables onto the O/R Designer so you can get a better idea of how the tool creates the classes. Drag the `ProductInventory` and `Location` tables from the Database Explorer onto the design surface. Once they are

on the surface, you can drag them around with the mouse to reposition them for better viewing. The design surface will now look similar to Figure 4-32.

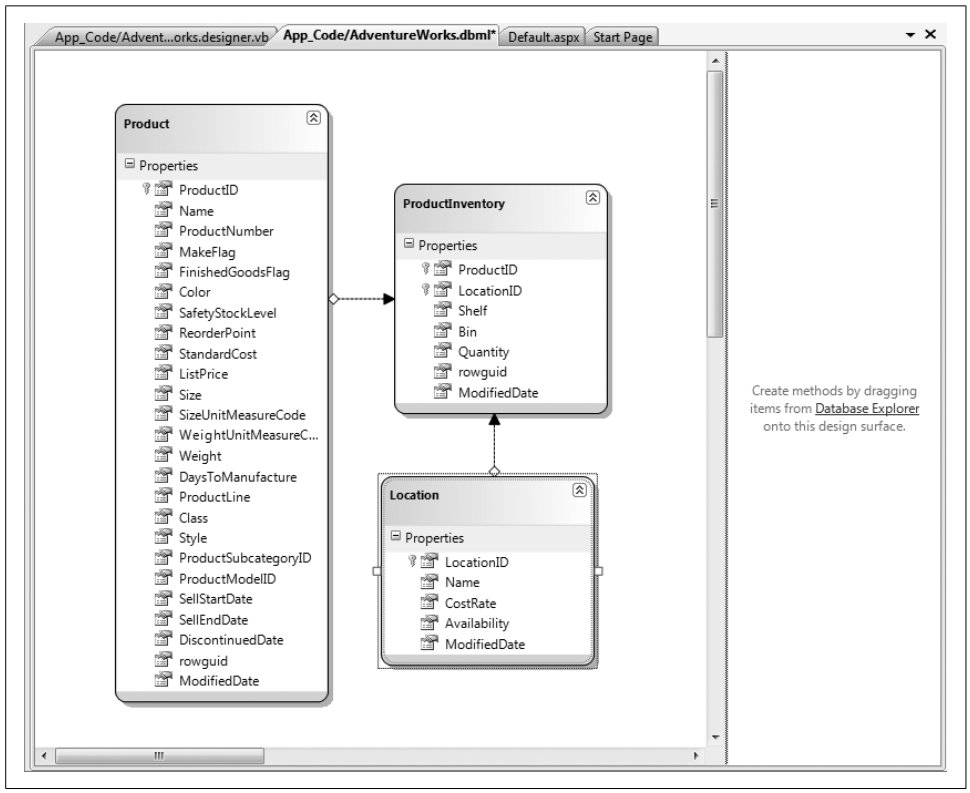


Figure 4-32. *After you drag multiple tables onto the O/R Designer from the Database Server, you can see the relationships between the tables.*

Because the database defined relationships between these tables, those relationships are reflected in the visual data model. More importantly, those relationships are now reflected in the underlying classes created by the tool.

We are not going to go into great detail about the underlying files, but it is very instructive to see how they are constructed. The *.dbml* file is an XML file which contains the structure of the tables in the O/R Designer. Normally, double-clicking this file opens up the visual designer, as shown in Figures 4-31 and 4-32. To see the actual XML code, close the visual designer, right-click on the *.dbml* file, select Open With…, and select XML Editor. You will see that the XML defines a `<Table>` element for each table. Within each `<Table>` element are a number of `<Column>` elements. Also, if there are any relationships with other tables, they are also defined within the `<Table>` element as `<Association>` elements.

The *.layout* file is used by the tool to keep track of the visual location and other design aspects of each table on the design surface. It does not have any real bearing on the object model or its underlying classes.

The *.vb* file contains the real meat. We looked at this file briefly above to see that it included the class declaration. If you open it now, having added tables with relationships, you will see there is a lot more to it. There is a lot of boilerplate code, which makes it easy to get lost, but essentially, this file defines three classes associated within the `AdventureWorksDataContext` class, one for each of the tables. Within each of these classes, there is a public property for each column in the table. You can see that each property has the same data type as the corresponding column in the table.

Save all these files and close them.

Using the Object Model

Now that you have an object model created, let's use it with a `LinqDataSource` control to display data in a `GridView`. Switch to Design view of *Default.aspx*. Drag two controls onto the page from the Data section of the Toolbox—a `LinqDataSource` and a `GridView`.

Select the `LinqDataSource` control, click on the Smart Tag, and select Configure Data Source…. This will bring up the Configure Data Source wizard, which is very similar to the wizard you saw earlier in this chapter used for configuring a `SqlDataSource` control. The first step in the wizard asks you to specify the context object, as shown in Figure 4-33.

All the available `DataContext` objects are available to choose from in the drop-down. In this case, there is only one, the `AdventureWorksDataContext` you created above, so click Next.

The next step asks you to select the table and fields, as shown in Figure 4-34.

Just as you did with the `SqlDataSource` back in Figure 4-8, select the `Products` table and the `ProductID`, `Name`, `ProductNumber`, `MakeFlag`, `SafetyStockLevel` and `ReorderPoint` columns. Then, click the Finish button.

Next, select the `GridView`, click on the Smart Tag, and then Choose Data Source. You will see that one of the available data sources is `LinqDataSource1`, the `LinqDataSource` control you just configured. Select that.

Immediately, the `GridView` will redraw in Design view to show the columns you have configured the data source to return.

While the Smart Tag is open, check the Enable Paging and Enable Sorting checkboxes, then run the page. You will see something identical to Figure 4-13, with paging and sorting fully implemented, except it is based on the `LinqDataSource` rather than the `SqlDataSource`.

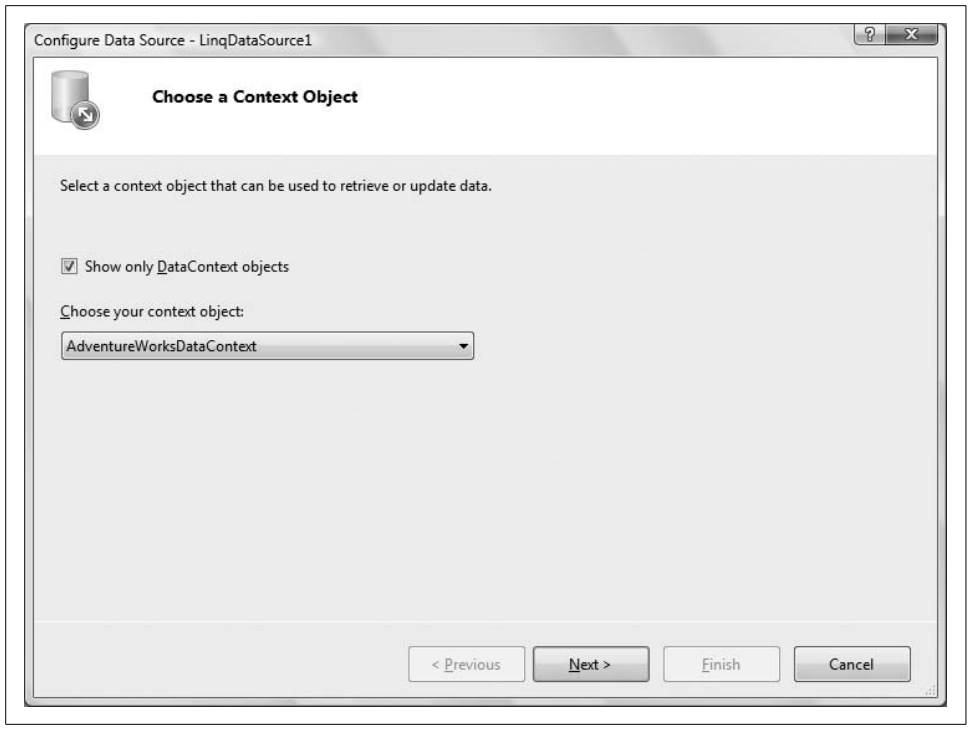

Configure Data Source - LinqDataSource1
Choose a Context Object
Select a context object that can be used to retrieve or update data.
Show only DataContext objects
Choose your context object:
AdventureWorksDataContext
< Previous
Next >
Finish
Cancel

Figure 4-33. *The first step in the Configure Data Source wizard for a LinqDataSource asks you to select a DataContext object.*

So, what is the difference between the two data sources, as the end result in these examples is identical? As we mentioned above, LINQ is a language feature that allows you to construct database queries directly in your language of choice, rather than using SQL. Back in Example 4-1, you saw the markup generated by the `SqlDataSource` control. It included a `ConnectionString` and a `SelectCommand`. The latter consists of this SQL statement:

```
SELECT ProductID, Name, ProductNumber, MakeFlag, SafetyStockLevel, ReorderPoint
FROM Production.Product
```

If you look at *Default.aspx* in Source view, you will see the following markup for the `LinqDataSource` control:

```
<asp:LinqDataSource ID="LinqDataSource1" runat="server"
      ContextTypeName="AdventureWorksDataContext"
      Select="new (ProductID, Name, ProductNumber, MakeFlag, SafetyStockLevel,
              ReorderPoint)"
      TableName="Products">
</asp:LinqDataSource>
```

Instead of a `ConnectionString` attribute pointing to a database, it has a `ContextTypeName` attribute specifying the `DataContext` class you created with the O/R

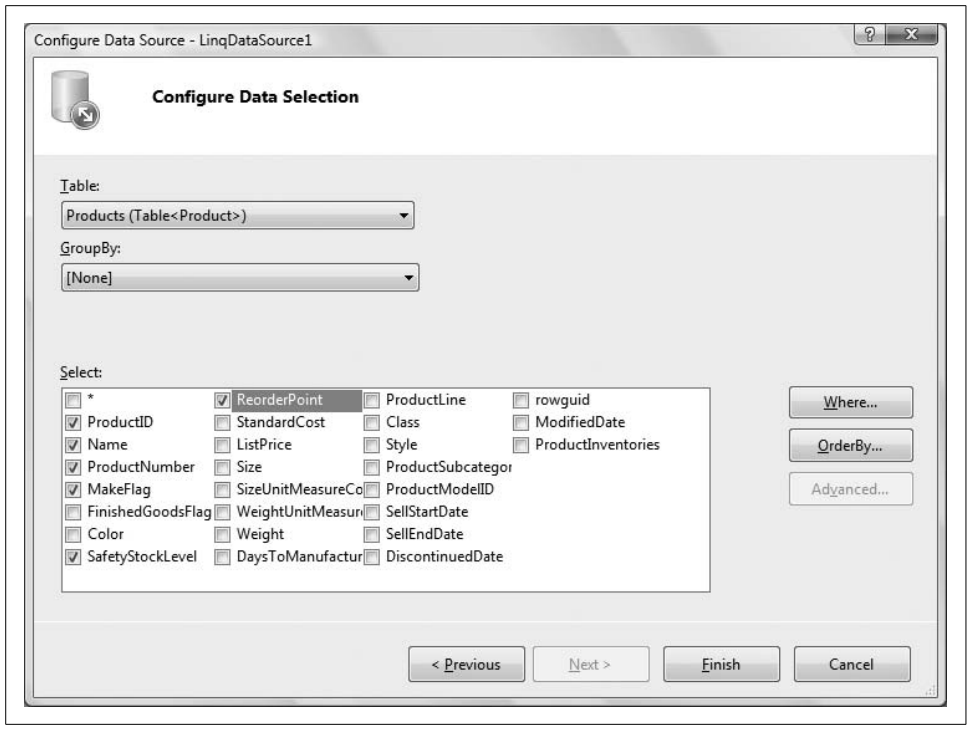


Figure 4-34. *The next step in the wizard asks you to select a table and fields to display.*

Designer. Instead of a `SelectCommand` attribute with a SQL statement, it has a `Select` attribute with a LINQ statement for selecting properties from the table class specified in the `TableName` attribute.

Just to reiterate the point that LINQ is a language feature, you could omit the `LinqDataSource` control from the page entirely and just construct your query directly in the VB code-behind.

To see this, add another `GridView` control to the page, which will be called `GridView2`. Do not choose a data source for this second `GridView`. Open the code-behind for the page and create a `Page_Load` method by selecting (`Page Events`) from the Class Name drop-down at the upper left of the editing surface and `Load` from the right drop-down. Then enter the highlighted code from Example 4-5.

Example 4-5. *Page_Load for databinding a GridView using LINQ statements*

```
Protected Sub Page_Load(ByVal sender As Object, _
   ByVal e As System.EventArgs) Handles Me.Load
   If Not IsPostBack then
      Dim dataContext As New AdventureWorksDataContext( )
      Dim products = From p In dataContext.Products _
                     Order By p.Name _
                     Select p.ProductID, p.Name, p.ProductNumber, p.MakeFlag, _
                        p.SafetyStockLevel, p.ReorderPoint
```

Example 4-5. *Page_Load for databinding a GridView using LINQ statements (continued)*

```
      GridView2.DataSource = products
      GridView2.DataBind( )
   End If
End Sub
```

Now when you run the page, you see that both grids display the same data. However, the second grid displays the columns in a different order, and it does not have paging and sorting enabled. You can fix the order of the columns by changing the markup for `GridView2` to set the `AutoGenerateColumns` attribute to `False` (the default is `True`) and add the same `BoundField` column declarations, as shown in the highlighted code in Example 4-6.

Example 4-6. *Markup for Default.aspx with two GridView controls, one populated from a LinqDataSource and one from the code-behind*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
   Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Untitled Page</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:LinqDataSource ID="LinqDataSource1" runat="server"
         ContextTypeName="AdventureWorksDataContext"
         Select="new (ProductID, Name, ProductNumber, MakeFlag, SafetyStockLevel,
               ReorderPoint)"
         TableName="Products">
      </asp:LinqDataSource>
   </div>
   <asp:GridView ID="GridView1" runat="server" AllowPaging="True" AllowSorting="True"
      AutoGenerateColumns="False" DataSourceID="LinqDataSource1">
      <Columns>
         <asp:BoundField DataField="ProductID" HeaderText="ProductID"
            ReadOnly="True" SortExpression="ProductID" />
         <asp:BoundField DataField="Name" HeaderText="Name" ReadOnly="True"
            SortExpression="Name" />
         <asp:BoundField DataField="ProductNumber" HeaderText="ProductNumber"
            ReadOnly="True" SortExpression="ProductNumber" />
         <asp:CheckBoxField DataField="MakeFlag" HeaderText="MakeFlag"
            ReadOnly="True" SortExpression="MakeFlag" />
         <asp:BoundField DataField="SafetyStockLevel" HeaderText="SafetyStockLevel"
            ReadOnly="True" SortExpression="SafetyStockLevel" />
         <asp:BoundField DataField="ReorderPoint" HeaderText="ReorderPoint"
            ReadOnly="True" SortExpression="ReorderPoint" />
      </Columns>
   </asp:GridView>
```

Example 4-6. *Markup for Default.aspx with two GridView controls, one populated from a LinqDataSource and one from the code-behind (continued)*

```
    <asp:GridView ID="GridView2" runat="server" AutoGenerateColumns="False">
        <Columns>
            <asp:BoundField DataField="ProductID" HeaderText="ProductID"
                ReadOnly="True" />
            <asp:BoundField DataField="Name" HeaderText="Name" ReadOnly="True" />
            <asp:BoundField DataField="ProductNumber" HeaderText="ProductNumber"
                ReadOnly="True" />
            <asp:CheckBoxField DataField="MakeFlag" HeaderText="MakeFlag"
                ReadOnly="True" />
            <asp:BoundField DataField="SafetyStockLevel" HeaderText="SafetyStockLevel"
                ReadOnly="True" />
            <asp:BoundField DataField="ReorderPoint" HeaderText="ReorderPoint"
                ReadOnly="True" />
        </Columns>
    </asp:GridView>
    </form>
</body>
</html>
```

Notice that `GridView2` does not have paging and sorting enabled, and the `BoundColumns` do not have a `SortExpression` specified. You could set the `AllowPaging` and `AllowSorting` attributes and include the `SortExpressions` in the Column declarations. If you did that, `GridView2` would appear to support paging and sorting, but an error would result when you tried to actually page or sort the grid. This is because the `LinqDataSource` automatically handles those chores for you. If you are binding directly from your code-behind without benefit of the `LinqDataSource` control, then you must write your own code to handle the paging and sorting events. This is not terribly difficult, but it is a bit beyond the scope of this book.

Editing Data in LINQ

The `LinqDataSource` control can also work with the `GridView` to easily allow for editing of your data, as long as the data source is configured to return all the columns of the table. The `GridView` does not have to display all the columns, but the `LinqDataSource` must select all the columns.

To see this, create a new web site called *LinqDataSourceEditing*. Just as you did in the previous example, add a LINQ to SQL Classes item to the web site and call it `AdventureWorks`. Allow it to be placed in the *App_Code* folder. Open the Database Explorer (Server Explorer in VS) and drag the Product table onto the left side of the O/R Designer.

Then, open the page, *Default.aspx*, in Source or Design view, and drag a `LinqDataSource` control and a `GridView` control onto the page from the Data section of the Toolbox. In Design view, open the Smart Tag of the `LinqDataSource` control and click on Configure Data Source…. As before, verify that AdventureWorksDataContext is chosen and click Next.

By default, the only table available, Products, should already be selected and the first column checkbox with an asterisk in it (to select all the fields) should be checked, as shown in Figure 4-35.

If no tables or fields are visible in the dialog, Cancel the dialog and click Build → Build Website to build the web site. Then try again.

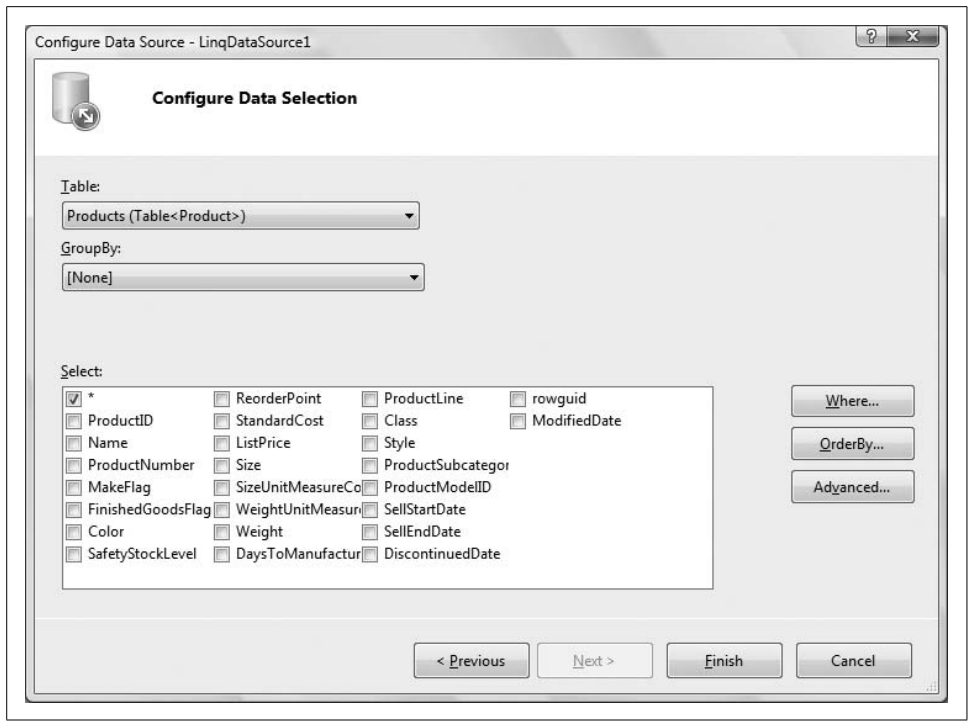


Figure 4-35. *To select all the fields of a table, check the asterisk.*

Click on the Advanced button to get the options shown in Figure 4-36.

Check all three checkboxes, then click OK, and then Finish. The Smart Tag for the control will now show checkboxes, all checked, for enabling Deletes, Inserts, and Updates, as shown in Figure 4-37.

If you look at the declaration for the `LinqDataSource` control in Source view, you will see the code in Example 4-7.

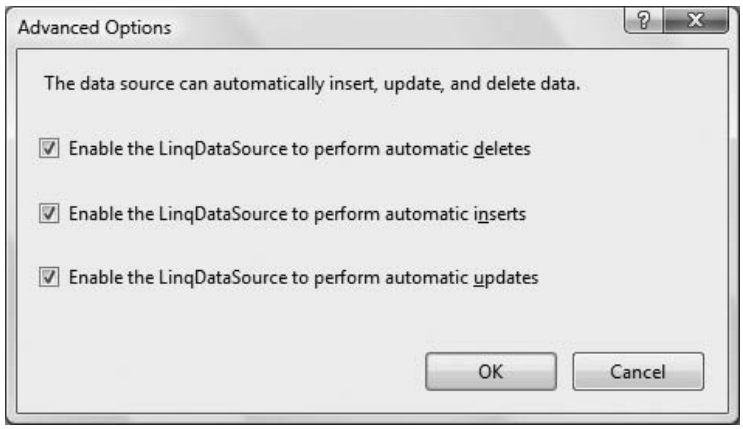

Advanced Options
The data source can automatically insert, update, and delete data.
Enable the LinqDataSource to perform automatic deletes
Enable the LinqDataSource to perform automatic inserts
Enable the LinqDataSource to perform automatic updates
OK
Cancel

Figure 4-36. *When configuring the LinqDataSource, clicking on the Advanced button allows you to configure advanced options.*

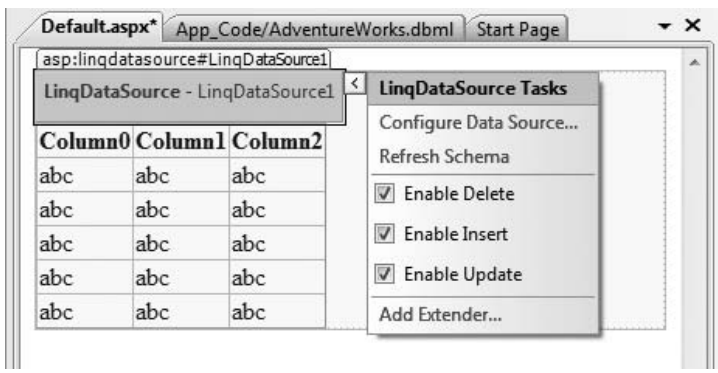


Figure 4-37. *After configuring the LinqDataSource, you can enable Deletes, Inserts, and Updates.*

Example 4-7. *Declaration for a LinqDataSource which enables editing*

```
<asp:LinqDataSource ID="LinqDataSource1" runat="server"
   ContextTypeName="AdventureWorksDataContext" EnableDelete="True"
   EnableInsert="True" EnableUpdate="True" TableName="Products">
</asp:LinqDataSource>
```

If you compare this with the equivalent declaration in Example 4-6, you will see that not only does this have attributes to enable Deletes, Inserts, and Updates, but it does not have a `Select` attribute returning specific columns, so all the columns are returned from the database.

Now, click on the Smart Tag for the `GridView`. Choose the `DataSource` to be `LinqDataSource1`. The `GridView` will immediately redraw with every column in the table, which is more than we want to display.

There are two ways to trim the unwanted fields. The UI way is to click on Edit Columns in the Smart Tag, to get the Fields editor, which you saw back in Figure 4-14. You can delete the unwanted fields by selecting them, one at a time, from the list box in the lower left corner of the dialog and clicking the red X button.

The easy way to get rid of the unwanted fields is to switch into Source view and just delete the undesired `BoundField` declarations from within the `Columns` element. So do that, removing all the `BoundField` declarations except for the ProductID, Name, ProductNumber, MakeFlag, SafetyStockLevel, and ReorderPoint columns, so that it will match the previous example.

Now, one last step: go back to the Smart Tag for the GridView. It will now have two new checkboxes in addition to the checkboxes you saw previously—Enable Editing and Enable Deleting. Check those, as shown in Figure 4-38.

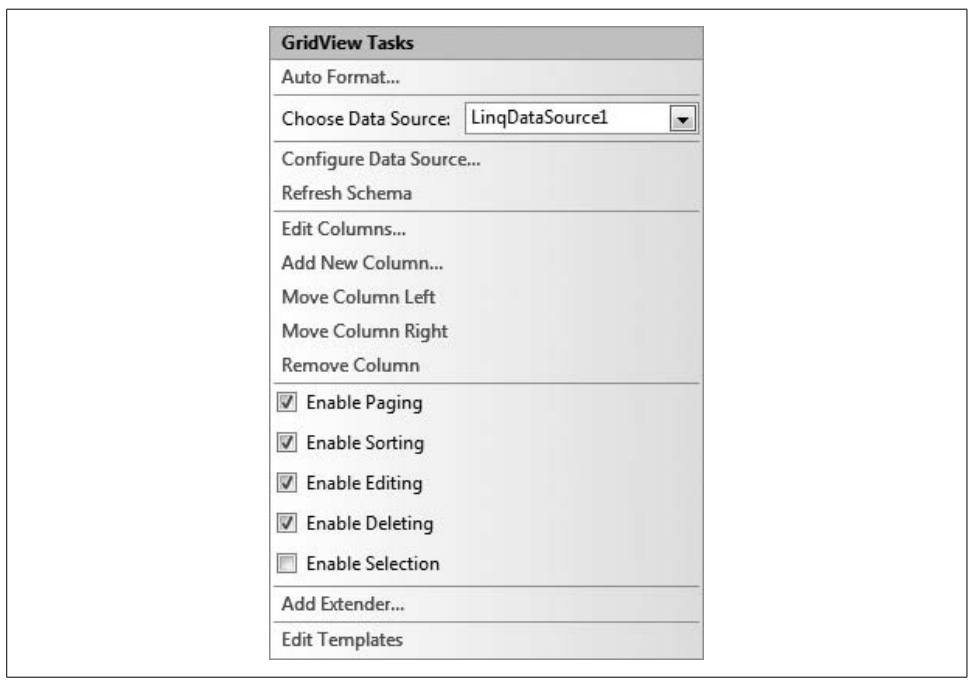


Figure 4-38. *The GridView Smart Tag now enables editing and deleting.*

Run the web site. You will get a grid that is functionally equivalent to that shown back in Figure 4-19, which you created with a `SqlDataSource`. The difference in appearance is only because we formatted that earlier grid and did not bother to do so with this latest example.

Just as you saw with the `SqlDataSource` example earlier in the chapter, if you try to delete a record from the Products table, you will get a database error caused by the fact that there are other tables with dependencies on the Products table. If you were allowed to delete products, it would leave orphan records in other tables. To deal with this, you would construct custom queries that first deleted the dependent records, then delete the product record. However, this is beyond the scope of this book.

ASP.NET Dynamic Data

Dynamic Data is a new feature of ASP.NET 3.5, introduced in Service Pack 1. It allows you to create a fully functional website implementing the four CRUD operations—Create, Read, Update, and Delete—with very little effort and virtually no handcoding. Depending on the requirements of the site you are building, this may save a lot of development effort.

The most obvious benefit of Dynamic Data is that it makes it easy and very fast to create a web site that lets users display and edit data, as seen in the example in this section. But Dynamic Data is more fundamentally about linking the presentation layer (how users view and work with data in web pages) with the data model (how the data is managed and stored).

In data-based applications, the database defines the type of data that can be store—for example, it defines whether a column consists of text, numbers, or dates, and it defines information such as how many characters a column value can contain. In addition, it defines how the data in one table relates to data in another table (that is, what relationships the tables have).

In most web sites that work with data, much of this information *about* the data has to be duplicated in the web site code. What data should be displayed in this page? How should the data be formatted? How can users edit this data? How should user input be validated? How does the data in a page work with related data in another page? These are all questions that web developers traditionally have to think about and to solve in web page after web page, and all by adding controls or code by hand to each individual page. The database already understands the data that it contains, yet the same information must be recoded in the web application by the web site developer.

To eliminate this type of redundancy, Dynamic Data makes the presentation layer "smart." Web pages can infer information about the data from the data layer at runtime—that is, dynamically. The result is the behavior that you see in this section. For example, web pages can automatically display data appropriately. Boolean data (true/false) is automatically displayed by using a check box, foreign keys are automatically displayed so that they link to related tables, and so on. Similarly, when a web page lets users edit or insert data, Dynamic Data infers the appropriate validation tests, based

directly on what the database allows. This ability to infer information and behavior from the data is sometimes referred to as *scaffolding*, and pages that can take advantage of this behavior are sometimes referred to as *scaffolded pages*.

Because every application is different, Dynamic Data is built so that you can customize its built-in behavior. And if that isn't sufficient, you can extend Dynamic Data to include new ways of interacting with data. Dynamic Data is a new feature of ASP.NET (in fact, it is still being developed), so the full potential of the feature has not yet been explored. But it has great promise, as we show you here.

This feature builds on the Object Relational Designer that you saw with LINQ, and the object model (data model) that it creates for you. It combines the object model with the extensive use of *templates* to produce fully editable pages representing any or all of the tables in your database, including all of the relationships between the tables.

Templates are controls or pages that define how users can interact with data. Field templates are used to define the look and behavior of individual data types from the data model. For example, a Text field template defines how string data is displayed; a TextEdit field template defines how users can edit and insert string data. Additional field templates are available for Boolean data, date/time data, and so on, and for displaying and working with foreign keys. Page templates define how sets of data (or tables) are displayed. There are page templates for simply listing data, for displaying master/detail data, for presenting editable data, and so on. Dynamic Data uses these templates automatically to display whatever data you are working with.

Dynamic Data is a big and complex topic, especially when you begin to customize how things are presented. We will present here a very simple example just to give you a feel for what is possible.

In VS/VWD, create a new web site. In the New Web Site dialog box, select the Dynamic Data Web Site template. Call the new web site DynamicAdventureWorks.

In the Solution Explorer, you will see a folder in the web site called DynamicData, which contains four subfolders that contain the templates. These are shown, somewhat expanded, in Figure 4-39.

Re-create the object model used in the two previous LINQ examples, adding the Product, ProductInventory, and Location tables to the object model. Recall from the discussion above that the O/R Designer will create a class called `AdventureWorksDataContext`. You will refer to this class in a moment.

Now comes the only coding you must do. Among the other files created for you by the IDE when creating a Dynamic Data web site is one called *global.asax* in the main folder of the web site. Double-click on this file in the Solution Explorer to open that file in the editor.

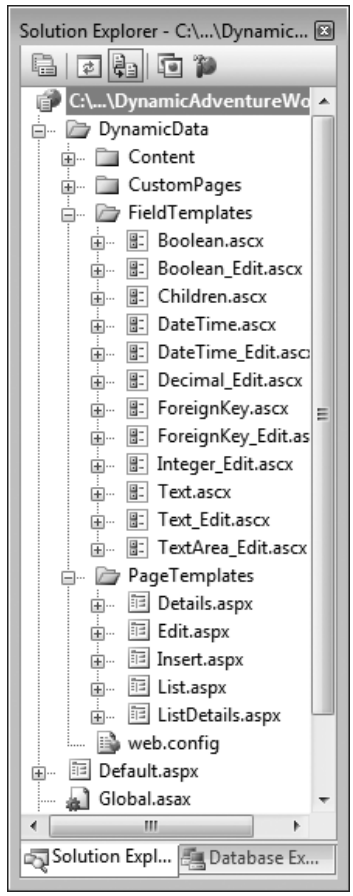


Figure 4-39. *A Dynamic Data web site has a folder called DynamicData which contains all the templates.*

The first method in *global.asax* is called `RegisterRoutes`, which initially looks like Example 4-8.

Example 4-8. *The original global.asax before, you make any edits*

```
Public Shared Sub RegisterRoutes(ByVal routes As RouteCollection)
    Dim model As New MetaModel

    '                    IMPORTANT: DATA MODEL REGISTRATION
    ‘ Uncomment this line to register LINQ to SQL classes or an ADO.NET Entity Data
    ' model for ASP.NET Dynamic Data. Set ScaffoldAllTables = true only if you are sure
    ' that you want all tables in the data model to support a scaffold (i.e. templates)
    ' view. To control scaffolding for individual tables, create a partial class for
    ' the table and apply the [Scaffold(true)] attribute to the partial class.
```

Example 4-8. *The original global.asax before, you make any edits (continued)*

```
    ' Note: Make sure that you change "YourDataContextType" to the name of the data
    ' context class in your application.
    ' model.RegisterContext(GetType(YourDataContextType), New ContextConfiguration() _
        With {.ScaffoldAllTables = False})

    ' The following statement supports separate-page mode, where the List, Detail, Insert,
    ' and Update tasks are performed by using separate pages. To enable this mode,
    ' uncomment the following route definition, and comment out the route definitions
    ' in the combined-page mode section that follows.
    routes.Add(New DynamicDataRoute("{table}/{action}.aspx") With { _
        .Constraints = New RouteValueDictionary(New With {.Action = _
        "List|Details|Edit|Insert"}), _
        .Model = model})

    ' The following statements support combined-page mode, where the List, Detail, Insert,
    ' and Update tasks are performed by using the same page. To enable this mode,
    ' uncomment the following routes and comment out the route definition in the
    ' separate-page mode section above.
    'routes.Add(New DynamicDataRoute("{table}/ListDetails.aspx") With { _
    '    .Action = PageAction.List, _
    '    .ViewName = "ListDetails", _
    '    .Model = model})

    'routes.Add(New DynamicDataRoute("{table}/ListDetails.aspx") With { _
    '    .Action = PageAction.Details, _
    '    .ViewName = "ListDetails", _
    '    .Model = model})
End Sub
```

You are going to modify one line in *global.asax*, the highlighted line in Example 4-8. In the original, this is actually a single, long commented line, but you are going to uncomment it in a moment, so we inserted a line continuation character to prepare it to wrap for readability.

Uncomment this highlighted line and make two edits. First, replace `YourDataContextType` with `AdventureWorksDataContext`. Second, change the value of `ScaffoldAllTables` from `False` to `True`. (Scaffolding is disabled by default because it adds some overhead to the web site, so you only want to enable it when you want to use Dynamic Data with all pages in the web site.) The line should now look like the following:

```
    model.RegisterContext(GetType(AdventureWorksDataContext), _
        New ContextConfiguration() With {.ScaffoldAllTables = True})
```

That's it. In Solution Explorer, right-click *Default.aspx*, and then click View in Browser. You will get a page like that shown in Figure 4-40.

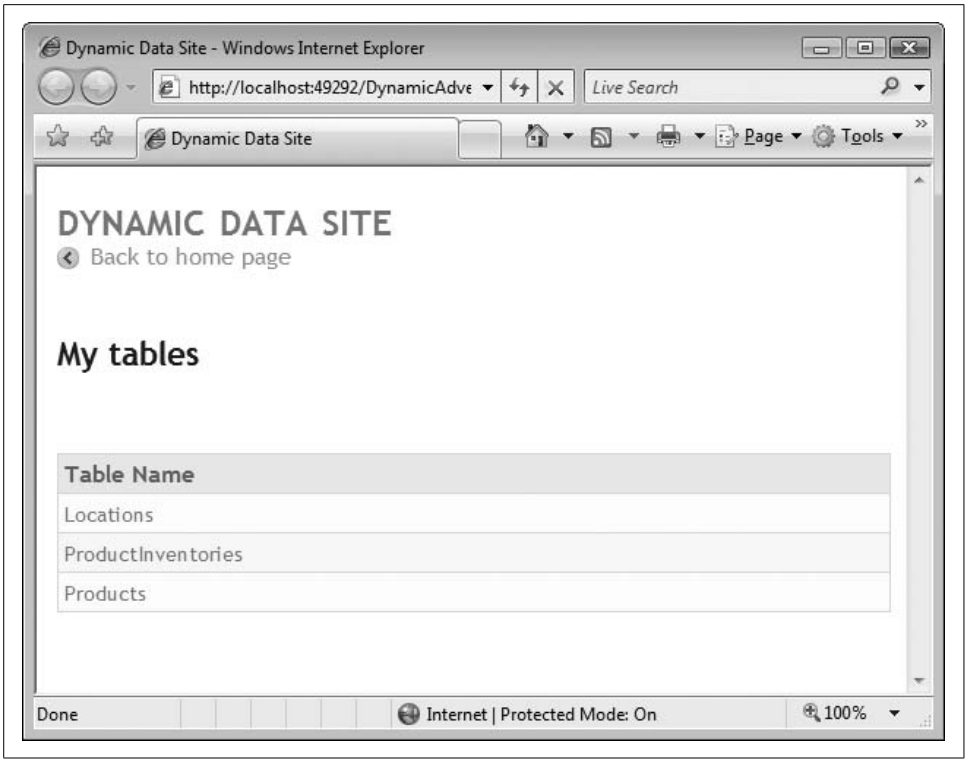


Figure 4-40. *This Dynamic Data web site was created from templates with virtually no handcoding.*

There are links for each of the tables you put in your object model. Clicking on any of these links brings up the data for that table, as shown in Figure 4-41.

Each page automatically includes a huge amount of functionality. Drop-downs at the top of the page allow you to filter the records displayed. You can click on any column heading to sort the table by that column. The first column includes links for editing, deleting, or selecting the row. Clicking on the Edit link allows you to edit the row with appropriate controls for the type of data (a `TextBox` for text data, for example, or a `CheckBox` for a `Boolean` data), and then either accept or cancel the edits. The Delete link deletes the row, and the Details link displays details about the row on a separate page. Also, not visible in this figure, the bottom of the grid has links for paging through the data. Links are also provided to rows in related tables, although they are also not shown in this figure.

All of this is totally customizable and extendable by editing the template files in the DynamicData folder.

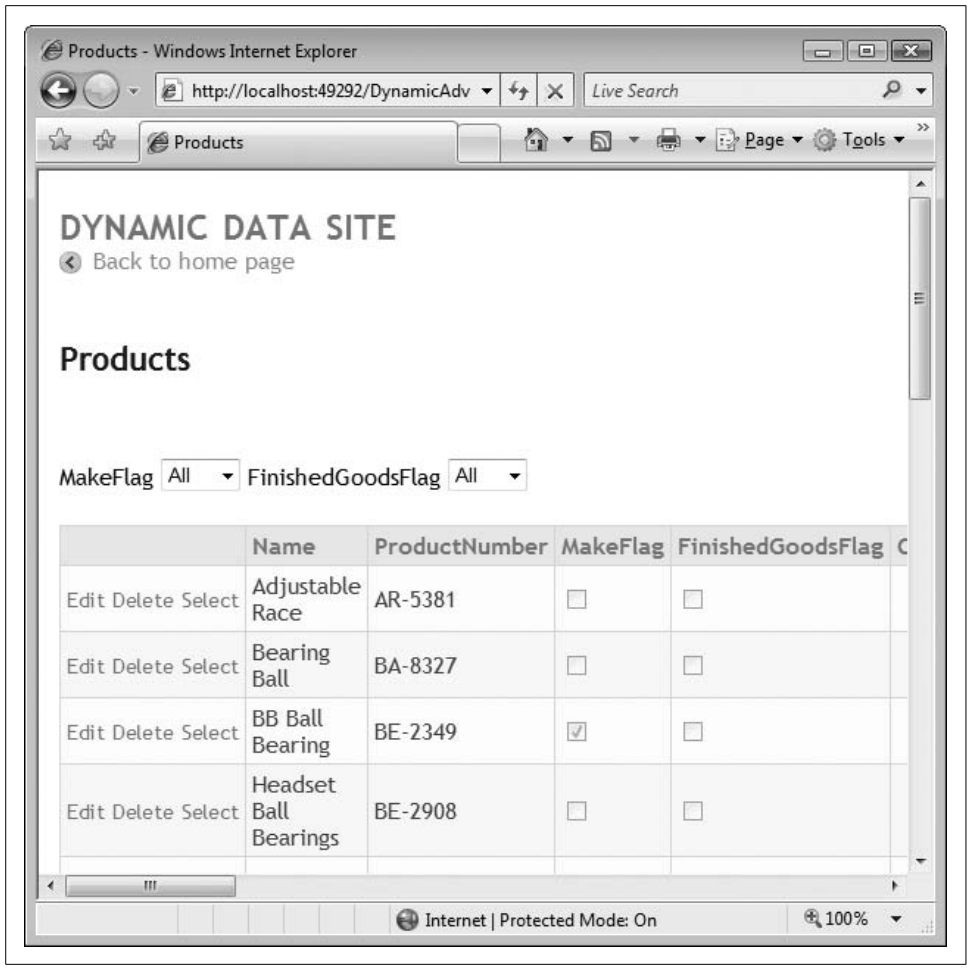


Figure 4-41. *The dynamically generated page for the Products table showing command links in the first column to Edit, Delete, or Select a row, and filter drop-downs for all the Boolean fields.*

Source Code Listings

The complete markup for the *Default.aspx* file in the AWProductData site is shown in Example 4-9, with the code-behind shown directly after in Example 4-10.

Example 4-9. *Default.aspx for AWProductData*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
      Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

Example 4-9. *Default.aspx for AWProductData (continued)*

```
   <title>Untitled Page</title>
</head>
<body>
   <form id="form1" runat="server">
   <asp:ScriptManager ID="ScriptManager1" runat="server">
   </asp:ScriptManager>
   <div>
      <asp:UpdatePanel ID="UpdatePanel1" runat="server">
         <ContentTemplate>
            <asp:SqlDataSource ID="SqlDataSource1" runat="server"
               ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString %>"

               SelectCommand="SELECT [ProductID], [Name], [ProductNumber], [MakeFlag],
                  [SafetyStockLevel], [ReorderPoint] FROM [Production].[Product]"
               DeleteCommand="DELETE FROM [Production].[Product]
                  WHERE [ProductID] = @ProductID"
              InsertCommand="INSERT INTO [Production].[Product] ([Name], [ProductNumber],
                  [MakeFlag], [SafetyStockLevel], [ReorderPoint])
                  VALUES (@Name, @ProductNumber, @MakeFlag, @SafetyStockLevel,
                  @ReorderPoint)"
               UpdateCommand="UPDATE [Production].[Product] SET [Name] = @Name,
                  [ProductNumber] = @ProductNumber, [MakeFlag] = @MakeFlag,
                  [SafetyStockLevel] = @SafetyStockLevel, [ReorderPoint] = @ReorderPoint
                  WHERE [ProductID] = @ProductID" >
               <DeleteParameters>
                  <asp:Parameter Name="ProductID" Type="Int32" />
               </DeleteParameters>
               <UpdateParameters>
                  <asp:Parameter Name="Name" Type="String" />
                  <asp:Parameter Name="ProductNumber" Type="String" />
                  <asp:Parameter Name="MakeFlag" Type="Boolean" />
                  <asp:Parameter Name="SafetyStockLevel" Type="Int16" />
                  <asp:Parameter Name="ReorderPoint" Type="Int16" />
                  <asp:Parameter Name="ProductID" Type="Int32" />
               </UpdateParameters>
               <InsertParameters>
                  <asp:Parameter Name="Name" Type="String" />
                  <asp:Parameter Name="ProductNumber" Type="String" />
                  <asp:Parameter Name="MakeFlag" Type="Boolean" />
                  <asp:Parameter Name="SafetyStockLevel" Type="Int16" />
                  <asp:Parameter Name="ReorderPoint" Type="Int16" />
               </InsertParameters>
            </asp:SqlDataSource>
            <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
               AllowSorting="True" AutoGenerateColumns="False"
               DataKeyNames="ProductID" DataSourceID="SqlDataSource1"
               BackColor="#DEBA84" BorderColor="#DEBA84" BorderStyle="None"
               BorderWidth="1px" CellPadding="3" CellSpacing="2">
               <FooterStyle BackColor="#F7DFB5" ForeColor="#8C4510" />
               <RowStyle BackColor="#FFF7E7" ForeColor="#8C4510" />
               <Columns>
                  <asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
```

Example 4-9. *Default.aspx for AWProductData (continued)*

```
                    ButtonType="Button" ShowSelectButton="True" />
                <asp:BoundField DataField="ProductID" HeaderText="ProductID"
                    InsertVisible="False"
                    ReadOnly="True" SortExpression="ProductID" />
                <asp:BoundField DataField="Name" HeaderText="Name"
                    SortExpression="Name" />
                <asp:BoundField DataField="ProductNumber"
                    HeaderText="ProductNumber" SortExpression="ProductNumber" />
                <asp:CheckBoxField DataField="MakeFlag"
                    HeaderText="MakeFlag" SortExpression="MakeFlag" />
                <asp:BoundField DataField="SafetyStockLevel"
                    HeaderText="SafetyStockLevel" SortExpression="SafetyStockLevel" />
                <asp:BoundField DataField="ReorderPoint"
                    HeaderText="ReorderPoint" SortExpression="ReorderPoint" />
            </Columns>
            <PagerStyle ForeColor="#8C4510" HorizontalAlign="Center" />
            <SelectedRowStyle BackColor="#738A9C" Font-Bold="True" ForeColor="White" />
            <HeaderStyle BackColor="#A55129" Font-Bold="True" ForeColor="White" />
        </asp:GridView>
        <asp:Label ID="Label1" runat="server" Text="Name"></asp:Label>
        <asp:TextBox ID="txtName" runat="server" ReadOnly="True"></asp:TextBox>
      </ContentTemplate>
    </asp:UpdatePanel>
  </div>
  </form>
</body>
</html>
```

Example 4-10. *Default.aspx.vb for AWProductData*

```
Partial Class _Default
    Inherits System.Web.UI.Page

Protected Sub GridView1_RowDataBound(ByVal sender As Object, _
            ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs) _
            Handles GridView1.RowDataBound
   If e.Row.RowType = DataControlRowType.DataRow Then
      Dim cellProductNumber As TableCell = e.Row.Cells(3) ' ProductNumber column
      If cellProductNumber.Text.Substring(0, 2) = "CA" Then
          cellProductNumber.ForeColor = Drawing.Color.Green
      End If

      Dim cellMakeFlag As TableCell = e.Row.Cells(4) ' MakeFlag column
      Dim cb As CheckBox = CType(cellMakeFlag.Controls(0), CheckBox)
      If cb.Checked Then
          e.Row.Cells(2).ForeColor = Drawing.Color.Red
      End If
   End If

End Sub

Protected Sub GridView1_SelectedIndexChanged(ByVal sender As Object, _
```

Example 4-10. *Default.aspx.vb for AWProductData (continued)*

```
      ByVal e As System.EventArgs) _
      Handles GridView1.SelectedIndexChanged
   If GridView1.SelectedRow.RowType = DataControlRowType.DataRow Then
      Dim cellName As TableCell = GridView1.SelectedRow.Cells(2) ' Name column
      txtName.Text = cellName.Text
   End If
End Sub
End Class
```

The complete markup for the *Default.aspx* file in the AWProductDataOrderDetails site is shown in Example 4-11, and the code-behind is shown in Example 4-12.

Example 4-11. *Default.aspx for AWProductDataOrderDetails*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb" Inherits="_
Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Untitled Page</title>
</head>
<body>
   <form id="form1" runat="server">
   <asp:ScriptManager ID="ScriptManager1" runat="server">
   </asp:ScriptManager>
   <div>
      <asp:UpdatePanel ID="UpdatePanel1" runat="server">
         <ContentTemplate>
            <asp:SqlDataSource ID="SqlDataSource1" runat="server"
               ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString %>"
               SelectCommand="SELECT [ProductID], [Name], [ProductNumber], [MakeFlag],
                  [SafetyStockLevel], [ReorderPoint] FROM [Production].[Product]"
               DeleteCommand="DELETE FROM [Production].[Product]
                  WHERE [ProductID] = @ProductID"
               InsertCommand="INSERT INTO [Production].[Product] ([Name],
                  [ProductNumber], [MakeFlag], [SafetyStockLevel], [ReorderPoint])
                  VALUES (@Name, @ProductNumber, @MakeFlag, @SafetyStockLevel,
                  @ReorderPoint)"
               UpdateCommand="UPDATE [Production].[Product] SET [Name] = @Name,
                  [ProductNumber] = @ProductNumber, [MakeFlag] = @MakeFlag,
                  [SafetyStockLevel] = @SafetyStockLevel, [ReorderPoint] = @ReorderPoint
                  WHERE [ProductID] = @ProductID">
               <DeleteParameters>
                  <asp:Parameter Name="ProductID" Type="Int32" />
               </DeleteParameters>
               <UpdateParameters>
                  <asp:Parameter Name="Name" Type="String" />
                  <asp:Parameter Name="ProductNumber" Type="String" />
                  <asp:Parameter Name="MakeFlag" Type="Boolean" />
                  <asp:Parameter Name="SafetyStockLevel" Type="Int16" />
                  <asp:Parameter Name="ReorderPoint" Type="Int16" />
```

Example 4-11. *Default.aspx for AWProductDataOrderDetails (continued)*

```
            <asp:Parameter Name="ProductID" Type="Int32" />
         </UpdateParameters>
         <InsertParameters>
            <asp:Parameter Name="Name" Type="String" />
            <asp:Parameter Name="ProductNumber" Type="String" />
            <asp:Parameter Name="MakeFlag" Type="Boolean" />
            <asp:Parameter Name="SafetyStockLevel" Type="Int16" />
            <asp:Parameter Name="ReorderPoint" Type="Int16" />
         </InsertParameters>
      </asp:SqlDataSource>
      <asp:GridView ID="GridView1" runat="server"
         AllowPaging="True" AllowSorting="True"
         AutoGenerateColumns="False"
         DataKeyNames="ProductID" DataSourceID="SqlDataSource1"
         BackColor="#DEBA84" BorderColor="#DEBA84" BorderStyle="None"
         BorderWidth="1px" CellPadding="3" CellSpacing="2">
         <FooterStyle BackColor="#F7DFB5" ForeColor="#8C4510" />
         <RowStyle BackColor="#FFF7E7" ForeColor="#8C4510" />
         <Columns>
            <asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
               ButtonType="Button" ShowSelectButton="True" />
            <asp:BoundField DataField="ProductID" HeaderText="ProductID"
               InsertVisible="False"
               ReadOnly="True" SortExpression="ProductID" />
            <asp:BoundField DataField="Name" HeaderText="Name"
               SortExpression="Name" />
            <asp:BoundField DataField="ProductNumber" HeaderText="ProductNumber"
               SortExpression="ProductNumber" />
            <asp:CheckBoxField DataField="MakeFlag" HeaderText="MakeFlag"
               SortExpression="MakeFlag" />
            <asp:BoundField DataField="SafetyStockLevel"
               HeaderText="SafetyStockLevel" SortExpression="SafetyStockLevel" />
            <asp:BoundField DataField="ReorderPoint" HeaderText="ReorderPoint"
               SortExpression="ReorderPoint" />
         </Columns>
         <PagerStyle ForeColor="#8C4510" HorizontalAlign="Center" />
         <SelectedRowStyle BackColor="#738A9C" Font-Bold="True" ForeColor="White" />
         <HeaderStyle BackColor="#A55129" Font-Bold="True" ForeColor="White" />
      </asp:GridView>
      <asp:Label ID="Label1" runat="server" Text="Name"></asp:Label>
      <asp:TextBox ID="txtName" runat="server" ReadOnly="True"></asp:TextBox>
      <br />
      <asp:GridView ID="GridView2" runat="server"
         DataSourceID="AdventureWorksOrderDetails"
         BackColor="#DEBA84" BorderColor="#DEBA84" BorderStyle="None"
         BorderWidth="1px"
         CellPadding="3" CellSpacing="2">
         <FooterStyle BackColor="#F7DFB5" ForeColor="#8C4510" />
         <RowStyle BackColor="#FFF7E7" ForeColor="#8C4510" />
         <PagerStyle ForeColor="#8C4510" HorizontalAlign="Center" />
         <SelectedRowStyle BackColor="#738A9C" Font-Bold="True" ForeColor="White" />
         <HeaderStyle BackColor="#A55129" Font-Bold="True" ForeColor="White" />
```

Example 4-11. *Default.aspx for AWProductDataOrderDetails (continued)*

```
            </asp:GridView>
            <asp:SqlDataSource ID="AdventureWorksOrderDetails" runat="server"
               ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString %>"
               SelectCommand="SELECT [SalesOrderID], [CarrierTrackingNumber], [OrderQty],
                  [UnitPrice], [UnitPriceDiscount], [LineTotal]
                  FROM [Sales].[SalesOrderDetail]
                  WHERE ([ProductID] = @ProductID)
                  ORDER BY [SalesOrderID]">
               <SelectParameters>
                  <asp:ControlParameter ControlID="GridView1" Name="ProductID"
                     PropertyName="SelectedValue" Type="Int32" />
               </SelectParameters>
            </asp:SqlDataSource>
         </ContentTemplate>
      </asp:UpdatePanel>
   </div>
   </form>
</body>
</html>
```

Example 4-12. *Default.aspx.vb for AWProductDataOrderDetails*

```
Partial Class _Default
    Inherits System.Web.UI.Page

Protected Sub GridView1_RowDataBound(ByVal sender As Object, _
            ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs) _
            Handles GridView1.RowDataBound
   If e.Row.RowType = DataControlRowType.DataRow Then
      Dim cellProductNumber As TableCell = e.Row.Cells(3) ' ProductNumber column
      If cellProductNumber.Text.Substring(0, 2) = "CA" Then
          cellProductNumber.ForeColor = Drawing.Color.Green
      End If

      Dim cellMakeFlag As TableCell = e.Row.Cells(4) ' MakeFlag column
      Dim cb As CheckBox = CType(cellMakeFlag.Controls(0), CheckBox)
      If cb.Checked Then
          e.Row.Cells(2).ForeColor = Drawing.Color.Red
      End If
   End If

End Sub

Protected Sub GridView1_SelectedIndexChanged(ByVal sender As Object, _
      ByVal e As System.EventArgs) _
      Handles GridView1.SelectedIndexChanged
   If GridView1.SelectedRow.RowType = DataControlRowType.DataRow Then
      Dim cellName As TableCell = GridView1.SelectedRow.Cells(2) ' Name column
      txtName.Text = cellName.Text
   End If
End Sub
End Class
```

Summary

- Most useful web sites make use of a database. ASP.NET provides controls that make it easy to connect to a database, and retrieve and edit data.
- The `GridView` is the most commonly used control for displaying data, although there are others. The `GridView` can sort data, and present it in pages, for easy reading.
- Data controls need to be bound to a data source to display data. To do that, you provide a `DataSource` control, which connects to the database and retrieves the data.
- You configure a `DataSource` control using a wizard that allows you to set a connection string, and then helps you construct a SQL query for retrieving data, or you can enter your own custom query.
- You create a new connection with the Add Connection dialog, and then you can save it in your *web.config* file for future use.
- The SQL `SELECT` statement allows you to specify which columns of data you want to retrieve, and from which table. The Wizard can configure this statement for you automatically.
- The SQL `INSERT`, `UPDATE`, and `DELETE` statements allow you to add, edit, and remove data, respectively. The Wizard can also generate these statements for you automatically, and you can easily add buttons to perform these functions in your `GridView`.
- Optimistic concurrency is a technique that protects your data by only changing the database if no one else has changed it since you read the data. Again, the Wizard can enable optimistic concurrency for you.
- The `WHERE` SQL clause filters the data you retrieve by specifying a condition for the data. A row will only be retrieved if that condition is true.
- You can create event handlers for the `GridView`, which enables you to take action on rows as they're bound, and also allows you to take action on rows as they're selected.
- You can provide parameters to the `SELECT` query, which enables you to display data in a `GridView` based on the value of another control, even another `GridView`.
- Language Integrated Query (LINQ) is a new language capability that allows you to keep all of your code in one place, instead of separating it between the database and the page, and also allows you to treat data in databases as code objects that you can manipulate with object-oriented languages.
- LINQ maps the database to a set of classes. The easiest way to accomplish that mapping in VS/VWD is with the Object Relational Designer.
- The `DataContext` is an object that transfers data between the object model and the database itself.

- The `LinqDataSource` control provides access to LINQ, just as the `SqlDataSource` does for a SQL database.
- Adding a LINQ to SQL Classes item to your site adds a *.dbml* file to the *App_Code* folder for your site, which holds the structures you create in the O/R designer.
- The *.dbml* file is associated with a *.vb* or *.cs* file that contains the actual data classes.
- You configure the `LinqDataSource` control much the same as you would the `SqlDataSource` control using the Configure Data Source wizard.
- Behind the scenes, the `LinqDataSource` control uses the `ContextTypeName` attribute to connect to the `DataContext` class, and the `Select` attribute to specify properties to be selected from the table named in the `TableName` attribute.
- You can also use the `LinqDataSource` to edit the data in a `GridView`, but to do that, the `LinqDataSource` must select all the columns of the table, even if they're not displayed in the `GridView`.
- Dynamic Data, a feature of ASP.NET 3.5 Service Pack 1, links the presentation of data with the way that data is stored and managed.
- With Dynamic Data, web pages can determine, at runtime, the appropriate presentation for data stored in the database, a technique called scaffolding.
- Dynamic Data builds on the Object Model created by LINQ.
- Dynamic Data makes use of templates: field templates define the behavior of individual data types; page templates define the behavior of entire sets of data. You can customize these templates to fit your needs.

Adding the ability to access a database is arguably the most powerful improvement you can make to your site. It's easy see how accessing a database would make the Order Form site from previous chapters that much more useful. Even the best order form, though, can't retrieve the right data if users don't give it valid input—if they enter a four-digit zip code, for example, or an improperly formatted credit card number. The whole thing would work much more smoothly if there was a way to check that the user's responses are valid before you spend the time accessing the database. The good news is that ASP.NET provides such a way, called *validation*, and that's what you'll learn about in the next chapter.

Quiz

1. What type of control do you need to retrieve data from the database?
2. What is the name of the process for allowing a control, such as a `GridView`, to extract data from the retrieved tables and format it properly?
3. What is a connection string?
4. What are the four elements of CRUD?
5. How do you attach a data source to a `GridView`?
6. If your table has many rows, what should you do in the `GridView` to make it easier to read?
7. How can you enable users to change the contents of the database from your `GridView`?
8. How can you take an action based on the data in a row, as the table is loaded?
9. If you're using a using a `LinqDataSource` control connected to a `GridView`, and you want users to be able to edit the data in the database, how do you need to configure the data that the `LinqDataSource` retrieves?
10. What are templates, in terms of Dynamic Data?

Exercises

Exercise 4-1. We'll start out easy, letting you create your own `GridView`. Create a new web site called Exercise 4-1. Add to it a `GridView` control that shows records from the `Product` table with a `Weight` greater than 100. The `GridView` should list the Product ID, Product Name, Product Number, Color, and List Price. The user should be able to update and delete records, sort by rows, and page through the content. Use the Professional formatting scheme to give it some style. The result should look like Figure 4-42.

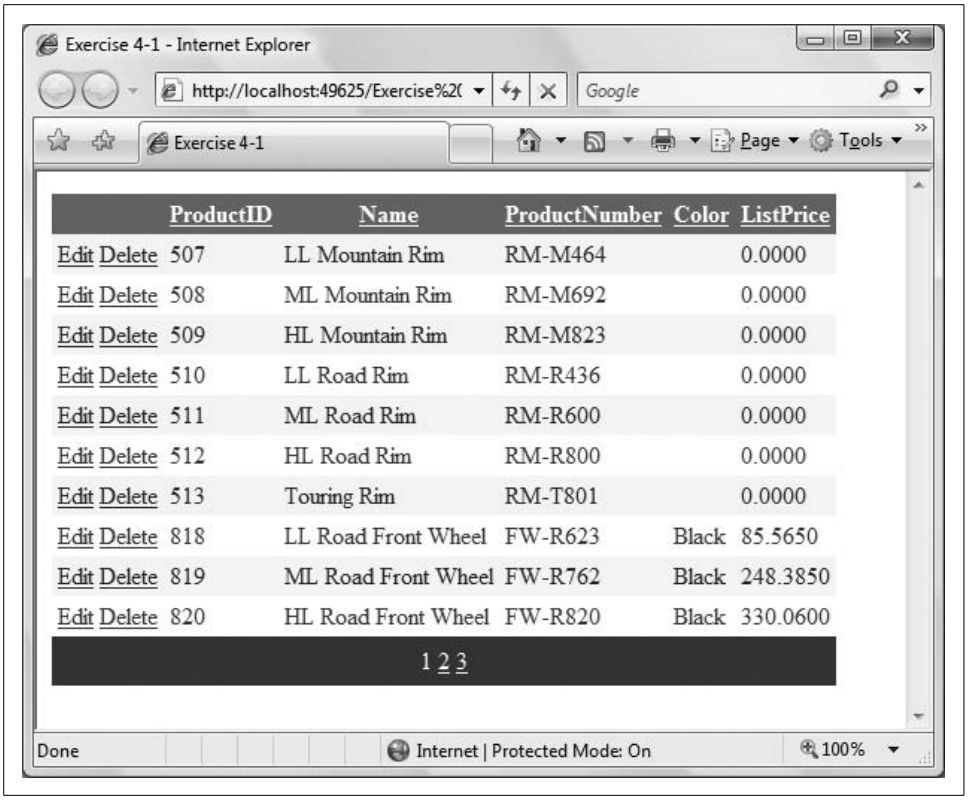

Exercise 4-1 - Internet Explorer
http://localhost:49625/Exercise%2(
Google
Exercise 4-1
Page
Tools
ProductID Name ProductNumber Color ListPrice
Edit Delete 507 LL Mountain Rim RM-M464 0.0000
Edit Delete 508 ML Mountain Rim RM-M692 0.0000
Edit Delete 509 HL Mountain Rim RM-M823 0.0000
Edit Delete 510 LL Road Rim RM-R436 0.0000
Edit Delete 511 ML Road Rim RM-R600 0.0000
Edit Delete 512 HL Road Rim RM-R800 0.0000
Edit Delete 513 Touring Rim RM-T801 0.0000
Edit Delete 818 LL Road Front Wheel FW-R623 Black 85.5650
Edit Delete 819 ML Road Front Wheel FW-R762 Black 248.3850
Edit Delete 820 HL Road Front Wheel FW-R820 Black 330.0600
1 2 3
Done
Internet | Protected Mode: On
100%

Figure 4-42. *Your goal for Exercise 4-1.*

Exercise 4-2. This one is a little trickier, but it lets you see how users could interact with the data in a `GridView`. Copy the web site from Exercise 4-1 to a new web site, called Exercise 4-2. Add the ability to select rows in your `GridView`. Add two labels and two read-only textboxes below the `GridView` to show the selected item's Product Name and color. The result should look like Figure 4-43.

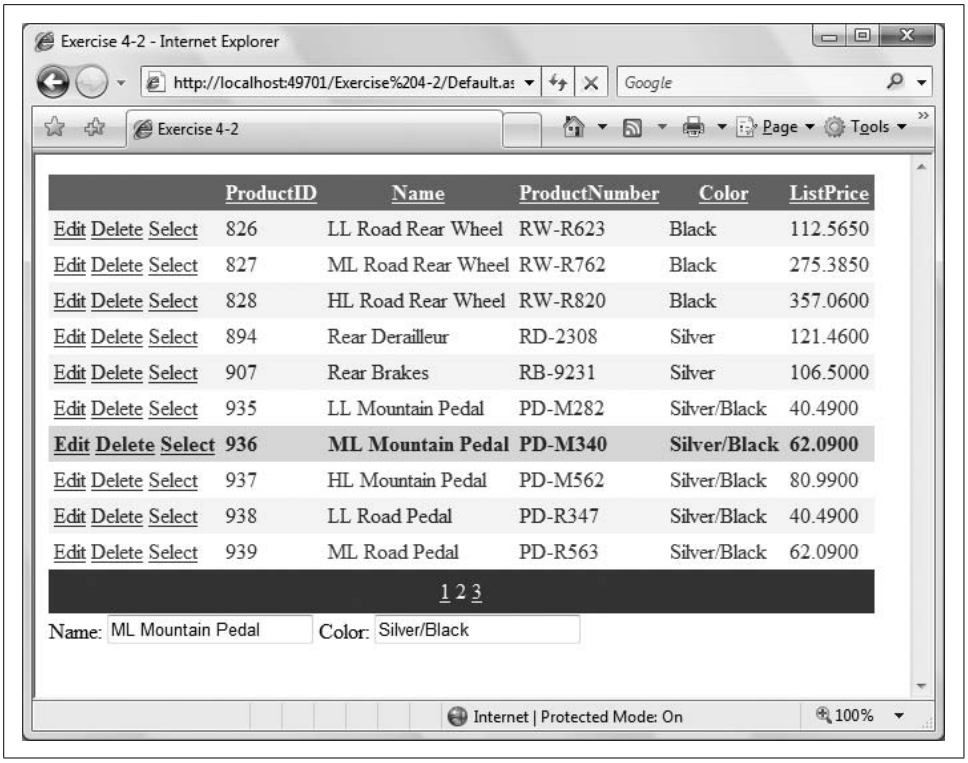

Exercise 4-2 - Internet Explorer
http://localhost:49701/Exercise%204-2/Default.as
Google
Exercise 4-2
Page
Tools
ProductID
Name
ProductNumber
Color
ListPrice
Edit Delete Select 826 LL Road Rear Wheel RW-R623 Black 112.5650
Edit Delete Select 827 ML Road Rear Wheel RW-R762 Black 275.3850
Edit Delete Select 828 HL Road Rear Wheel RW-R820 Black 357.0600
Edit Delete Select 894 Rear Derailleur RD-2308 Silver 121.4600
Edit Delete Select 907 Rear Brakes RB-9231 Silver 106.5000
Edit Delete Select 935 LL Mountain Pedal PD-M282 Silver/Black 40.4900
Edit Delete Select 936 ML Mountain Pedal PD-M340 Silver/Black 62.0900
Edit Delete Select 937 HL Mountain Pedal PD-M562 Silver/Black 80.9900
Edit Delete Select 938 LL Road Pedal PD-R347 Silver/Black 40.4900
Edit Delete Select 939 ML Road Pedal PD-R563 Silver/Black 62.0900
1 2 3
Name: ML Mountain Pedal
Color: Silver/Black
Internet | Protected Mode: On
100%

Figure 4-43. *Your goal for Exercise 4-2.*

Exercise 4-3. Now it's time to combine what you've learned from previous chapters with the new stuff, and throw a little AJAX into the mix as well. Create a new web site called Exercise 4-3. This site should have a radio button that gives readers the opportunity to select whether they want to see data from the Employee table or the Customer table. Use two `Panel` controls. The Employee panel should have a `GridView` showing the EmployeeID, ManagerID, and Title. The Customer panel should have a `GridView` showing the Customer ID, Account Number, and Customer Type. The table that the reader chooses should appear dynamically in a new panel; the other one should be invisible. The result should look like Figure 4-44.

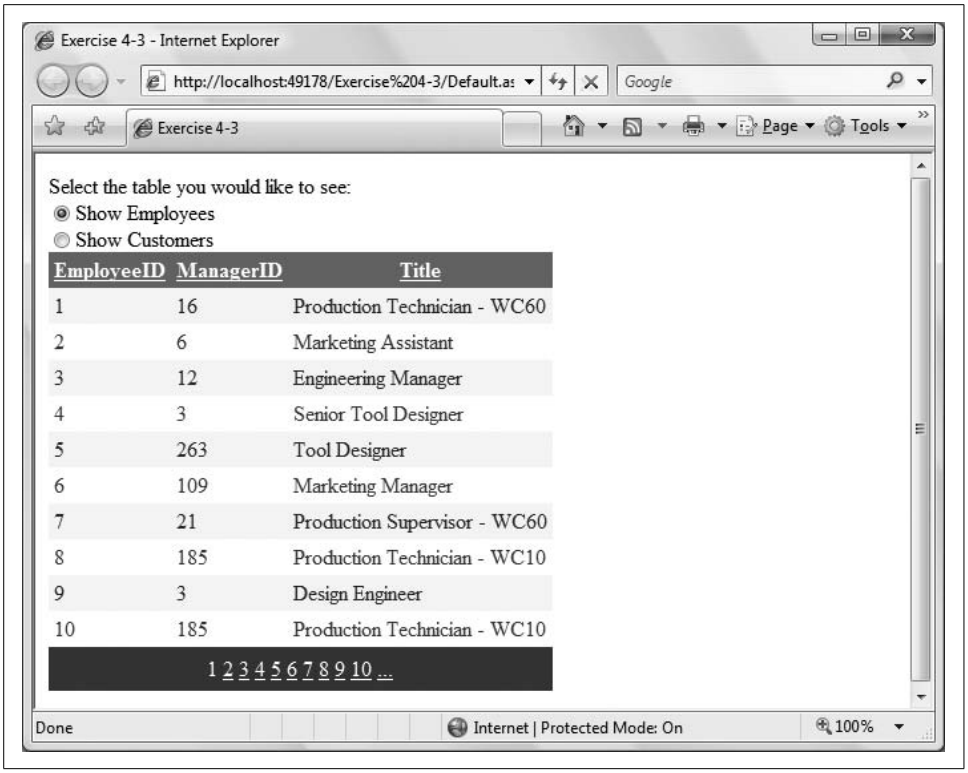


Figure 4-44. *Your goal for Exercise 4-3.*

Exercise 4-4. Ready for a bit of a challenge? Sure you are. You're going to see how to retrieve data based on multiple customer selections—like you would in a shopping site. Create a new web site called Exercise 4-4. This site should have three drop-down menus:

- A Category drop-down list that lists the product categories from the `ProductCategory` table
- A Subcategory drop-down list that lists the subcategories of the Category listed in the first drop-down, by using the `ProductSubcategory` table
- A Color drop-down list that lists the available product colors from the `Product` drop-down list

In addition, there should be a Submit button that users click. Below all of this is a `GridView` that displays the Products (from the Product table) that match the chosen subcategory and color. (You don't need to match the category—all that control does is dictate the contents of the Subcategory table.) The `GridView` should display the ProductID, Name, Product number, and the color, just so you can tell it's working.

(Hint: You can use the `DISTINCT` SQL statement to avoid duplication in the grid.) It should look like Figure 4-45. Note that many of the possible combinations in the drop-down lists won't produce any results in the grid. You can test your solution with the choices shown in Figure 4-45.

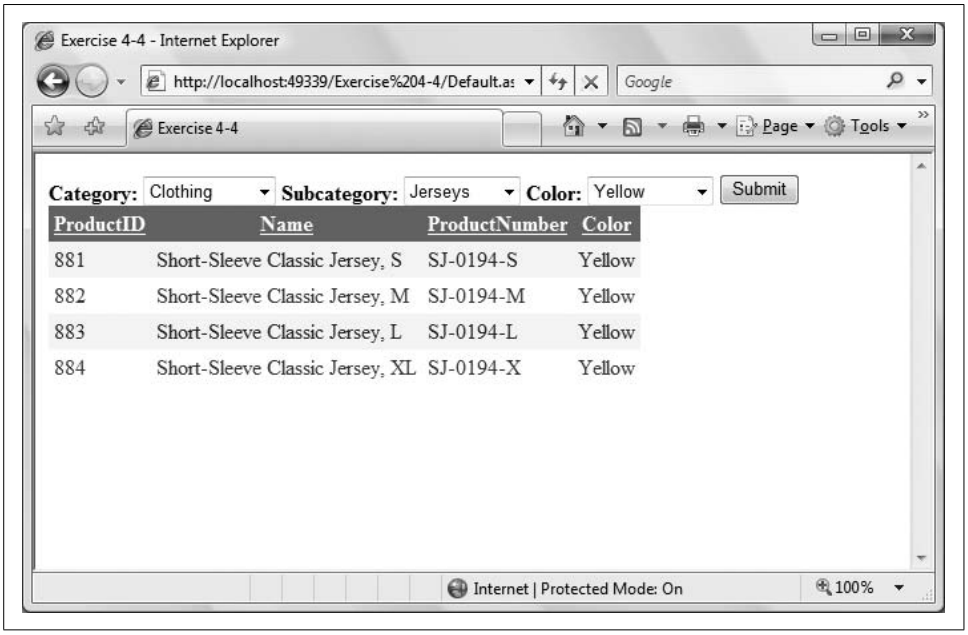


Figure 4-45. *Your goal for Exercise 4-4.*

CHAPTER 5
Validation

As you have seen in the preceding chapters, many web applications involve user input. The sad fact is, however, that users make mistakes: they skip required fields, they put in six-digit phone numbers, and they return all manner of incorrectly formatted data to your application. Your database routines can choke on corrupted data, and orders can be lost. An incorrectly entered credit card number or omitted address may result in a missed sales opportunity. Fortunately, you can write code that checks the user's input before it gets anywhere near your database code, or anything else dangerous. The process of verifying the user's input is called *validation*.

Traditionally, it takes a great deal of time and effort to write reliable validation code. You need to check each field and create routines for ensuring data integrity. If bad data is found, you need to display an error message so the user knows there is a problem and how to correct it.

In any given application, you may choose to verify that the data is formatted correctly, or that the values fall within a given range, or that certain fields have a value at all. For example, if you're processing an order, you may need to ensure that the user has input an address and phone number, that the phone number has the right number of digits (and no letters), and that the Social Security number entered is in the appropriate form of nine digits separated by hyphens.

Some applications require more complex validation, in which you validate that one field is within a range established by two other fields. For example, you might ask in one field what date the customer wishes to arrive at your hotel, and in a second field you might ask for the departure date. When the user books a dinner reservation, you'll want to ensure that the date the user chooses is between the arrival and departure dates.

There is no limit to the complexity of the validation routines you may need to write. Credit cards have checksums built into their values, as do ISBN numbers. Zip and postal codes follow complex patterns, as do international phone numbers. You may need to validate passwords, membership numbers, dollar amounts, dates, runway choices, or launch codes.

In addition, you usually want all of this validation to happen client side so you can avoid the delay of repeated round trips (postbacks) to the server while the user is tinkering with his input. In the past, this was solved by writing client-side JavaScript to validate the input, and then writing server-side script to handle input from browsers that don't support client-side programming. In addition, as a security check, you may want to do server-side validation even though you have validation implemented in the browser, as users can circumvent client-side validation code by creating a malicious page that masquerades as a legitimate page (a tactic known as *spoofing*). Typically, these security measures involved writing your validation code twice, once for the client and once for the server.

As you can see, in traditional web programming, validation requires extensive custom programming. The ASP.NET framework simplifies this process by providing rich controls for validating user input. In addition to checking the validity of the data entered, the validation controls allow you to specify how and where error messages will be displayed: either inline with the input controls, aggregated in a summary report, or both. You can use these controls to validate input for HTML and ASP.NET server controls. In this chapter, you'll learn how to use all these validation controls, and you'll create a number of pages that you can adapt right away to use on your site.

Validation Controls

You add validation controls to your ASP.NET document as you would add any other control. As a property of the validation control, you specify which other control you're validating. You may freely combine the various validation controls, and you may even write your own custom validation controls, as you'll see later in this chapter.

With current browsers that support DHTML, .NET validation is done on the client side, avoiding the necessity of a round trip to the server. (This client-side validation uses JavaScript but is not part of the AJAX library.) With older browsers, *your* code is unchanged, but the code sent to the client ensures validation at the server.

Validation occurs whenever the page tries to post back to the server. Sometimes you don't want any validation to occur, such as when a Cancel button is clicked. To prevent validation in these circumstances, many postback controls—such as `Button`, `ImageButton`, `LinkButton`, `ListControl`, and `TextBox`—have a `CausesValidation` property, which you can set to dictate whether validation is performed on the page when the control's default event is raised.

If `CausesValidation` is set to `true`, the default value—the postback—will *not* occur if any control on the page fails validation. This is a big deal, because it means the page will not post to the server unless all of the controls on the page are in a valid state. If `CausesValidation` is set to `false`, however, no validation will occur when that button is used to post the page.

Sometimes you need a postback to be allowed to proceed even if some controls on the page are invalid. For example, suppose you have a page that gathers both address and tax information. A button on the page processes the address fields, which might be before the user has entered some unrelated tax information. However, if a required tax field is missing, the page will not post.

You solve this problem by using the `ValidationGroup` property. You can group a bunch of validation controls together with the control (or controls) that causes the postback so that only validation controls that are members of the group will be applied. In this example, you can require that all the address controls are valid before allowing the Address button to post, but allow the post even if some tax fields are invalid.

ASP.NET includes the following validation controls:

`RequiredFieldValidator`
: Ensures the user does not leave the field blank and skip over your input control. A `RequiredFieldValidator` can be tied to a text box, which means that the page will only pass validation if the user enters something into the text box. With selection controls, such as a drop-down or radio buttons, the `RequiredFieldValidator` ensures the user makes a selection other than the default value you specify. The `RequiredFieldValidator` does not examine the validity of the data; it only ensures that some data is entered or chosen.

`RangeValidator`
: Ensures that the value entered is within a specified lower and upper boundary. You can specify the range to be within a pair of numbers (such as greater than 10 and less than 100), a pair of characters (greater than D and less than K), or a pair of dates (after 1/1/08 and before 2/28/08).

`CompareValidator`
: Compares the user's entry against another value. It can compare against a constant you specify at design time, or against a property value of another control. It can also compare against a database value.

`RegularExpressionValidator`
: One of the most powerful validators, it compares the user's entry with a regular expression you provide. *Regular expressions* are a powerful way to match a pattern of letters, numbers, or symbols, as you'll see later in the chapter. You can use this validator to check for valid Social Security numbers, phone numbers, password strength, and so forth.

`CustomValidator`
: If none of these controls meets your needs, you can create your own using the `CustomValidator`. This checks the user's entry against whatever algorithm you provide in a custom method.

In the remainder of this chapter, we'll examine how to use each of these controls to validate data in ASP.NET applications.

The RequiredFieldValidator

The `RequiredFieldValidator` ensures the user provides a value for your control, or in the case of drop drop-down lists, that the user picks something other than the default value.

To get started, create a new web site called `RequiredFieldValidator`. In this section, you're going to create the shipping selection web page shown in Design view in Figure 5-1. This is a pretty standard shipping form, as you can see on any number of web sites. You can imagine how you'd incorporate such a page into your own site.

Although ASP.NET validation controls do much of their work client-side using JavaScript, they neither use nor depend on AJAX, including the `ScriptManager` control, so you don't need to include one for this page.

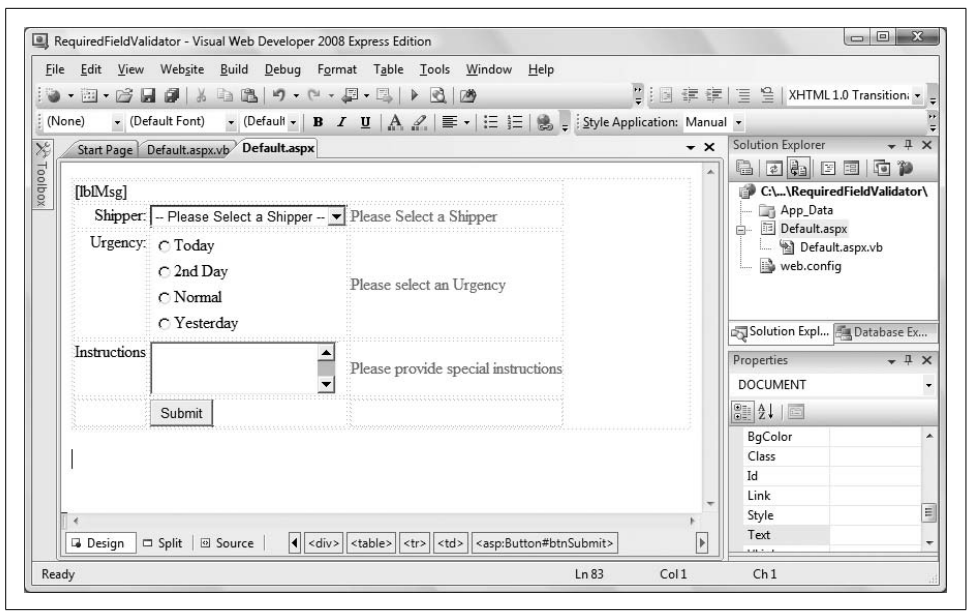


Figure 5-1. *This shipping selection page incorporates RequiredFieldValidator controls to ensure that users make selections in each field.*

When the user clicks the Submit button, the page is validated to ensure that each field has been modified. If not, the offending field is marked with an error message in red, as shown in Figure 5-2.

You'll use a five-row, three-column HTML table to create the page layout, into which you'll place the necessary controls. Using what you've learned from the previous chapters, you can create the table fairly easily, either directly in Source view, or using the Table Wizard, so we won't go over that here. Adding the controls to the form is also pretty easy, as you'll see.

The first column of the table contains some descriptive captions for the input fields, with the middle column containing the controls to be validated (a `DropDownList`, a `RadioButtonList`, and a `TextBox`). The third column will contain the validation controls and any error messages they want to display.

The first row of the table should be a single cell spanning all three columns (using the HTML `colspan` attribute) containing a `Label` for displaying any messages. Set the `ID` of that `Label` to `lblMsg`, and set the `Text` property to an empty string (`Text=""`).

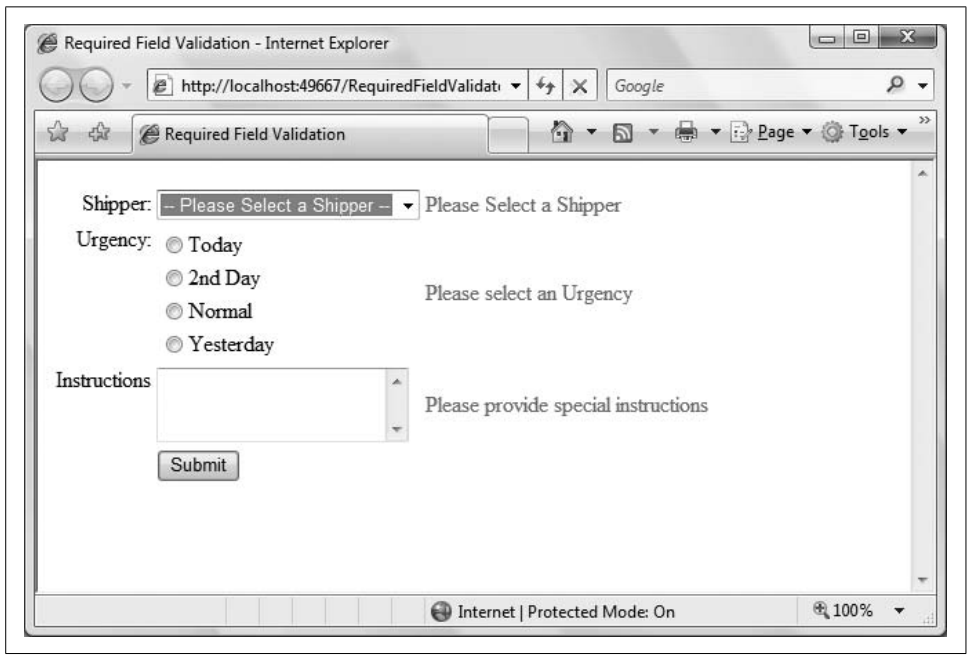


Figure 5-2. *The user of this page didn't provide any shipping information before clicking Submit, so the RequiredFieldValidator controls return errors, which you can see in the column on the right.*

The first row of user input controls has a `DropDownList` in the middle column named `ddlShipper`. Use the ListItem Collection Editor to set the values for the Shipper `DropDownList` to the following:

-- Please Select a Shipper --
US Postal Service
Overnight Express
United Shipping Service

WHL
Pony Express
Starship Transporter

The first item is particularly important, as you'll see in a minute.

For each field that you want validated, add a `RequiredFieldValidator` control, which is a control like any other. Open the Validation section of the Toolbox, and drag three `RequiredFieldValidator` controls into the table, into the third cell of each of the three middle rows, as shown previously in Figure 5-1.

The `RequiredFieldValidator` control has its own `ID`, and it also asks for the `ID` of the control you wish to validate. Therefore, set the `ID` of the first `RequiredFieldValidator` you added to `rfvShipper`, and set its `ControlToValidate` property to `ddlShipper` (the drop-down list that you are validating). Delete the text in the `ErrorMessage` property for the moment; you don't need it right now. You'll need to include some text to show to the user if he doesn't make a selection, though, so include the text "Please select a shipper" between the opening and closing tags of the control. You could also set this text using the `Text` property.

The `RequiredFieldValidator` has an additional attribute, `InitialValue`, which you should set to the initial value of the control being validated, in this case, the drop-down box. If the user clicks Submit, this initial value will be compared with the value of the drop-down, and if they are the same, the error message will be displayed. This forces the user to change the initial value. In this case, the first item in the drop-down is the bit of text asking the user to make a choice. That isn't a valid selection, so you need to make sure that the user chooses something else, by setting `InitialValue` to be the same as that text prompt. You *don't* want to use `InitialValue` if you have a default shipper, for example, because that would prevent the user from selecting the default. All this is shown in the following snippet in Source view:

```
<asp:RequiredFieldValidator ID="rfvShipper" runat="server"
   ControlToValidate="ddlShipper"
   InitialValue="-- Please Select a Shipper --">
   Please Select a Shipper
</asp:RequiredFieldValidator>
```

Make sure the `Display` attribute is set to `Static` (the default), which tells ASP.NET to allocate room on the page for the validator whether there is a message to display or not. If you set this to `Dynamic`, space will not be allocated until (and unless) an error message is displayed. Dynamic allocation is powerful, but it can cause your controls to bounce around on the page when the message is displayed. We'll show you how this looks in a minute.

In the second input row, add a `RadioButtonList` called `rblUrgency`, with the items shown previously in Figure 5-1. Give the second `RequiredFieldValidator` an `ID` of `rfvUrgency`, and set its `ControlToValidate` property to `rblUrgency`, to ensure that one

of the radio buttons in rblUrgency is selected. Also, change the Text property to "Please select an Urgency":

```
<asp:RequiredFieldValidator ID="rfvUrgency" runat="server"
   ControlToValidate="rblUrgency"
   Display="Static">
   Please select an Urgency
</asp:RequiredFieldValidator>
```

You do not need to indicate an initial value this time. Because the control is a radio button list, the validator knows the user is required to pick one of the buttons; if any button is chosen, the validation will be satisfied.

Finally, to complete the example, add a multiline text box in the third row and assign it an ID of txtInstructions. Use the third RequiredFieldValidator, named rfvInstructions, to require the user to enter some text in it. The validator is straightforward; set the text box as ControlToValidate, and enter the error message "Please provide special instructions" into the Text property, for display if the box is left empty:

```
<asp:RequiredFieldValidator ID="rfvInstructions" runat="server"
   ControlToValidate="txtInstructions"
   Display="Static">
   Please provide special instructions
</asp:RequiredFieldValidator>
```

In a real site, the Special Instructions field would probably be optional, but we'll make it required for this example.

In the last row, place a Button in the middle column, call it btnSubmit, and set its Text to "Submit". The only code required in the code-behind file is the event handler for the Submit button. Double-click on the button in Design view, and you'll be taken to the Click event handler, as usual. Enter the highlighted code from Example 5-1.

Example 5-1. *The Button Click event handler for RequiredFieldValidator checks the validation for every control on the page*

```
Protected Sub btnSubmit_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles btnSubmit.Click
   If Page.IsValid Then
      lblMsg.Text = "Page is valid!"
   Else
      '  this code never reached
      lblMsg.Text = "Some of the required fields are empty."
   End If
End Sub
```

When the Submit button is clicked, the validation for each control is checked, and if every control is valid, the `IsValid` property of the page will return `true`.

Now, go back and set the `Display` property of all the validation controls to `Dynamic`. Run the application and see what happens. No space is allocated for the validation controls, and the browser will consider your table to be only two columns wide rather than three. That is, the table will not allocate any space for the validation messages, and will recognize only one column for the prompt and the other for the controls. When you validate the controls (by clicking the Submit button), the table will widen, which can be either disconcerting or attractive, depending on how you design the layout of the page. In this case, you can see the controls jump around when you click Submit, which probably isn't what you want, so after you've played with it a bit, go back and change the controls back to `Static`.

Take a look back at Example 5-1. Notice the comment that says the `else` clause will never be reached. Recall that the validation occurs client-side. If the page is not valid, it is never even posted to the server, and the server-side code does not run (unless, of course, you set the `CausesValidation` property to `false`, as described at the beginning of this chapter).

You can make your pages a bit friendlier for your users by placing the focus on the first control that fails validation. To do so, add the `SetFocusOnError` property to each validation control and set it to `true` (the default is `false`):

```
<asp:RequiredFieldValidator runat=server
   id="rfvInstructions"
   ControlToValidate="txtInstructions"
   Display="Static"
   SetFocusOnError="true"
   Width="100%" >
   Please provide special instructions
</asp:RequiredFieldValidator>
```

Run your application again, and click Submit without adding special instructions. After validation, you not only get the message asking you to provide them, but the focus is on the `TextBox` control, ready for you to enter text. If you set `SetFocusOnError` on more than one control, and if the page is invalid, the focus will be set to the first control that fails validation and has this property set to `true`.

The complete source code for the markup file, *default.aspx*, for this example is listed in Example 5-2.

Example 5-2. *Default.aspx forRequiredFieldValidator*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
   Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

Example 5-2. *Default.aspx forRequiredFieldValidator (continued)*

```
<head runat="server">
   <title>Required Field Validation</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <table>
         <tr>
            <td colspan="3">
               <asp:Label ID="lblMsg" runat="server" Text="" />
            </td>
         </tr>
         <tr>
            <td align="right" valign="top">Shipper:</td>
            <td>
               <asp:DropDownList ID="ddlShipper" runat="server">
                  <asp:ListItem>-- Please Select a Shipper --</asp:ListItem>
                  <asp:ListItem>US Postal Service</asp:ListItem>
                  <asp:ListItem>Overnight Express</asp:ListItem>
                  <asp:ListItem>United Shipping Service</asp:ListItem>
                  <asp:ListItem>WHL</asp:ListItem>
                  <asp:ListItem>Pony Express</asp:ListItem>
                  <asp:ListItem>Starship Transporter</asp:ListItem>
               </asp:DropDownList>
            </td>
            <td>
               <asp:RequiredFieldValidator ID="rfvShipper" runat="server"
                  ControlToValidate="ddlShipper"
                  InitialValue="-- Please Select a Shipper --"
                  SetFocusOnError="true">
                  Please Select a Shipper
               </asp:RequiredFieldValidator>
            </td>
         </tr>
         <tr>
            <td align="right" valign="top">Urgency:</td>
            <td >
               <asp:RadioButtonList ID="rblUrgency" runat="server">
                  <asp:ListItem>Today</asp:ListItem>
                  <asp:ListItem>2nd Day</asp:ListItem>
                  <asp:ListItem>Normal</asp:ListItem>
                  <asp:ListItem>Yesterday</asp:ListItem>
               </asp:RadioButtonList>
            </td>
            <td>
               <asp:RequiredFieldValidator ID="rfvUrgency" runat="server"
                  ControlToValidate="rblUrgency"
                  Display="Static"
                  SetFocusOnError="true">
                  Please select an Urgency
               </asp:RequiredFieldValidator>
            </td>
```

Example 5-2. *Default.aspx forRequiredFieldValidator (continued)*

```
        </tr>
        <tr>
           <td align="right" valign="top">Instructions</td>
           <td>
              <asp:TextBox ID="txtInstructions" runat="server"
                 Rows="3" TextMode="MultiLine"></asp:TextBox>
           </td>
           <td>
              <asp:RequiredFieldValidator ID="rfvInstructions" runat="server"
                 ControlToValidate="txtInstructions"
                 Display="Static"
                 SetFocusOnError="true">
                 Please provide special instructions
              </asp:RequiredFieldValidator>
           </td>
        </tr>
        <tr>
           <td></td>
           <td>
              <asp:Button ID="btnSubmit" runat="server" Text="Submit"  />
           </td>
           <td></td>
        </tr>
     </table>
  </div>
  </form>
</body>
</html>
```

The Summary Control

As you saw in the previous example, putting your validation feedback next to each control can be useful, but it leads to some possible layout problems. Fortunately, ASP.NET lets you decide how you want to report validation errors. For example, rather than putting error messages alongside the control, you can summarize all the validation failures with a `ValidationSummary` control. This control can place a summary of the errors in a bulleted list, a simple list, or a paragraph that appears elsewhere on the web page or in a pop-up message box.

To see how this works, create a copy of the `RequiredFieldValidator` web application called `RequiredFieldValidatorSummary`. Switch to Source view. From the Validation section of the Toolbox, drag a `ValidationSummary` control onto the bottom of the page, after the `</table>` tag.

The steps for copying a web site to a new web site are presented in Appendix B.

Set the attributes of this ValidationSummary control to the values highlighted in the following code snippet (you can do this in the Properties window as well, of course):

```
<asp:ValidationSummary ID="vsSummary" runat="server"
        DisplayMode="BulletList"
        HeaderText="The following error(s) were found:"
        ShowSummary="true" />
```

To make this work, you'll need to add an ErrorMessage attribute to the other validation controls. For example, modify the first validation control for the Shipper drop-down menu as follows:

```
<asp:RequiredFieldValidator ID="rfvShipper" runat="server"
   ControlToValidate="ddlShipper"
   Display="Static"
   InitialValue="-- Please Select a Shipper --"
   ErrorMessage="You did not select a shipper from the drop-down."
   SetFocusOnError="true">*
</asp:RequiredFieldValidator>
```

Note that the asterisk (*) is added after the opening tag for the validator, and not in the Text property this time. If this control reports a validation error, the text in the ErrorMessage attribute will be displayed in the summary. You've also modified the validator to display an asterisk rather than the more complete error message. Now that you have a summary, you don't need to put a complete error message by each control; you need only flag the error. Now, make similar changes for each of the other RequiredFieldValidator controls (you can use the error text displayed in Figure 5-3, or feel free to improvise).

Run your application, and click Submit without making any choices, so that none of the validation controls pass muster. The results are shown in Figure 5-3.

In Figure 5-3, the summary of validation errors is presented as a bulleted list. This is the default display mode, although we did set it explicitly. Messages can also be displayed as a simple list or a single paragraph by setting the DisplayMode property of the ValidationSummary to BulletList, List, or SingleParagraph, respectively.

The Compare Validator

While the ability to ensure the user has made some sort of entry is great, you will often want to validate that the entry content is within certain guidelines. A common requirement is to compare the user's input to a constant value, the value of another control, or a database value.

To see this at work, make a new copy of the RequiredValidationSummary web site and name the new web site CompareValidator. In this example, you'll add a new control that asks the user how many packages she expects to receive as part of a shipment.

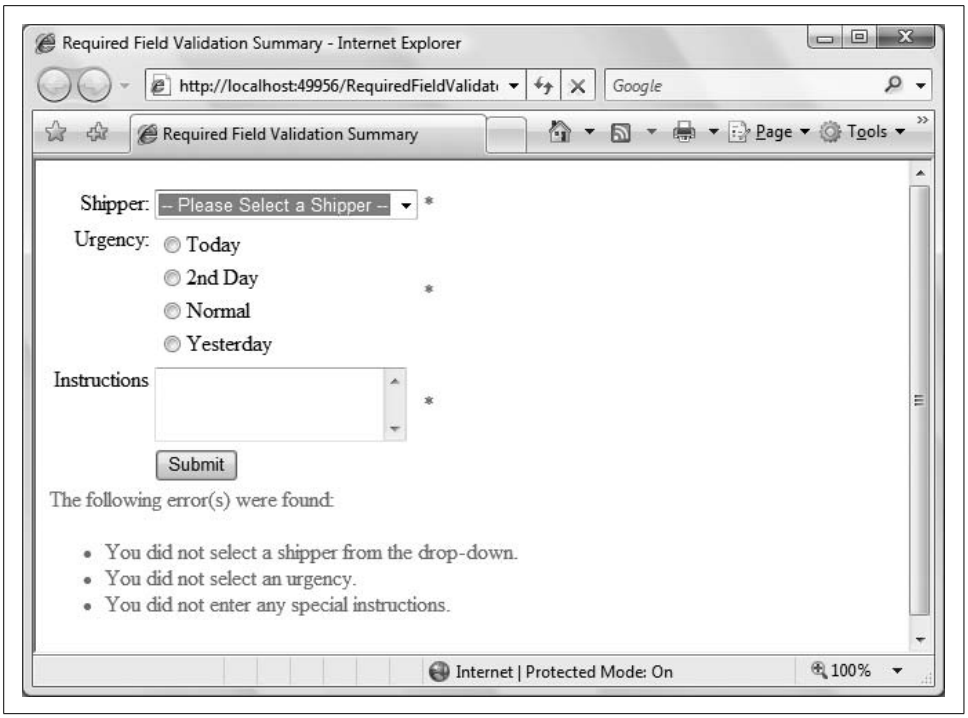


Figure 5-3. *When you use a Validation Summary, the controls that didn't pass validation are marked, but more importantly, the summary appears on the page telling the user exactly what's wrong.*

To do so, you'll first need to add a new table row before the row that contains the Submit button. You can do this in Source or Design view, although the latter makes it very easy.

Click anywhere in the row containing the Submit button. Then, right-click to get a pop-up menu and select Insert, then Row Above to insert a new table row above the current row, as shown in Figure 5-4.

This will insert a blank HTML table row above the selected row. In that new row, insert some caption text, a text box (call it `txtNumPackages`), a required field validator, and a compare validator into the correct table cells. You may want to switch to Source view to tweak the layout with a `<br/>` between the two validation controls. Adjust the properties for the two new validators as shown in Example 5-3.

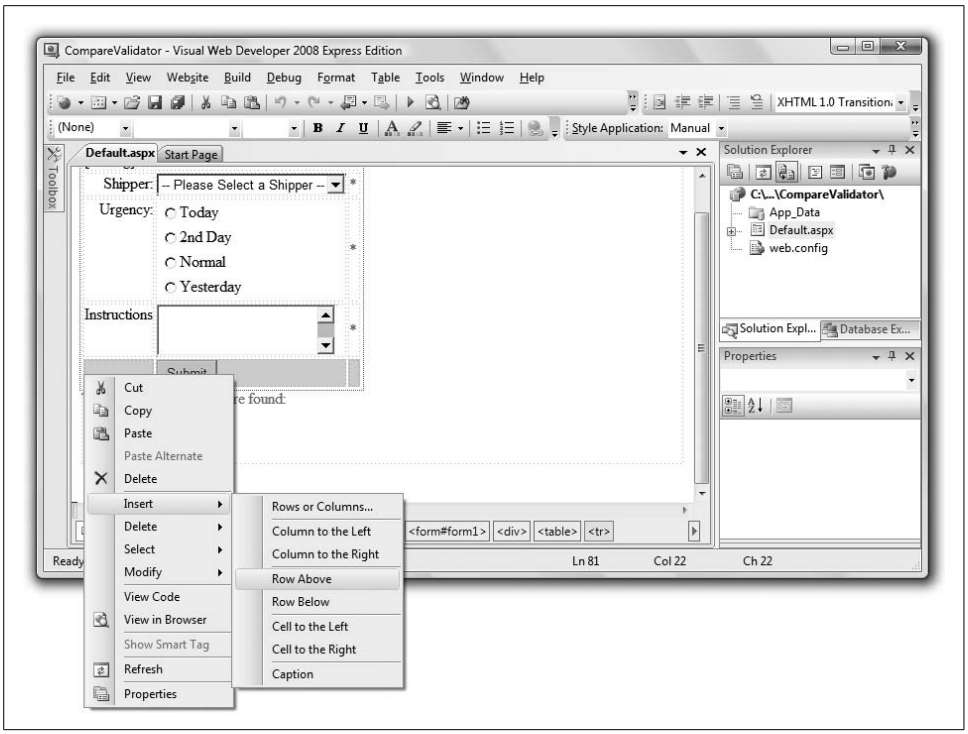


Figure 5-4. *When you select a table row and right-click, the pop-up menu gives you options for inserting a new row or cell where you want.*

Example 5-3. *The CompareValidator control compares a control to a set value*

```
<tr>
   <td>Number of packages</td>
   <td>
      <asp:TextBox ID="txtNumPackages" runat="server"></asp:TextBox>
   </td>
   <td>
      <asp:RequiredFieldValidator ID="rfvNumPackages" runat="server"
         ControlToValidate="txtNumPackages"
         ErrorMessage="You did not enter the number of packages."
         SetFocusOnError="True">*
      </asp:RequiredFieldValidator>
      <br />
      <asp:CompareValidator ID="cvNumPackages" runat="server"
         ControlToValidate="txtNumPackages"
         ErrorMessage="Invalid number of packages"
         Operator="GreaterThan" SetFocusOnError="True"
         Type="Integer" ValueToCompare="0">*
      </asp:CompareValidator>
   </td>
</tr>
```

The table now looks like Figure 5-5 in Design view, with the two validation controls in the last cell circled.

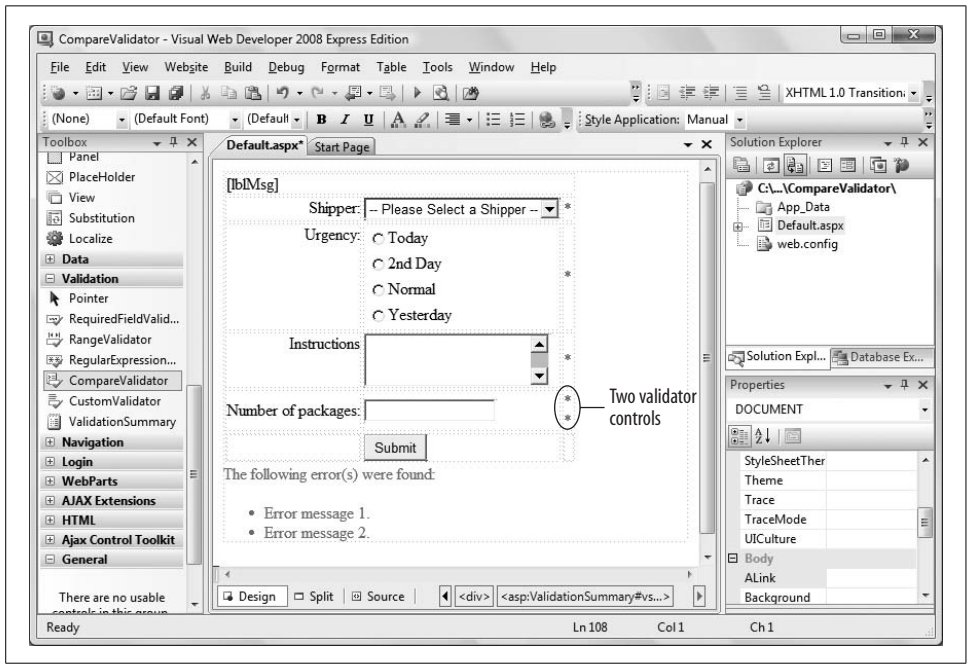


Figure 5-5. *A new table row with a TextBox to be validated, and the RequiredFieldValidator and CompareValidator validation controls in Design view.*

Run your application again, and try entering various values into the field. You'll see that if you enter 0, or a negative number, the validation fails. If you enter a positive number, the control passes validation. If you leave it blank, you'll notice that it still fails. Without the `RequiredFieldValidator`, though, it would have passed.

Both validators are placed into the same cell in the table, and both validators validate the same control: `txtNumPackages`. The `RequiredFieldValidator` is needed because the `CompareValidator` will always return `true` for null or empty values, as nonintuitive as that sounds.

The `CompareValidator`'s `ValueToCompare` attribute takes a constant; in this case, zero. The `Operator` attribute determines how the comparison will be made (that is, how the input value must be related to the `ValueToCompare`).

The possible values for the `Operator` attribute are `Equal`, `NotEqual`, `GreaterThan`, `GreaterThanEqual`, `LessThan`, `LessThanEqual`, and `DataTypeCheck`. In this example, to be valid, the input value must be greater than the `ValueToCompare` constant. Or to put in more relevant terms, the user must send more than zero packages.

You must use the Type attribute to tell the control what type of value it is using. The Type attribute takes one of the ValidationDataType enumerated values: Currency, Date, Double (a Double is VB-speak for a noninteger number, i.e., a decimal number), Integer, or String. In the example, the values are compared as integers, and thus, entering (for example) a character will cause the validation to fail.

Checking the Input Type

Rather than checking that the number of packages is greater than zero, you might want to check that it is a number at all (rather than a letter or date). To do this, you make a minor change to the CompareValidator.

Remove the ValueToCompare attribute and change the Operator attribute from GreaterThan to DataTypeCheck. Because the Type attribute is Integer, the control will report any integer value as valid. Use the following code to replace that for the CompareValidator you added in the last section:

```
<asp:CompareValidator ID="cvNumPackages" runat="server"
   ControlToValidate="txtNumPackages"
   ErrorMessage="Invalid number of packages"
   Operator="DataTypeCheck" SetFocusOnError="True"
   Type="Integer" >*
</asp:CompareValidator>
```

Now, run the application again, and try entering random data in the text box. You'll see that numbers, even zero or negative numbers, pass validation, while anything else fails. You can imagine how this sort of validation would be useful for order numbers, credit card numbers, or anyplace where the type of data is important.

Comparing to Another Control

You can compare a value in one control to the value in another control, rather than to a constant. A classic use of this might be to ask the user to enter his password twice and then validate that both entries are identical.

The common scenario is that you've asked the user to pick a new password. For security, when the password is entered, the text is disguised with asterisks. Because this will be the password the user will need to log in, you must validate that the user entered the password as intended, without errors. The typical solution is to ask the user to enter the password a second time, and then check that the same password was entered each time. The CompareValidator is perfect for this.

To demonstrate this, you'll need to add two table rows to your page, each with a TextBox for use as a password field. The first of these password fields will have a RequiredFieldValidator control; the second will have both a RequiredFieldValidator and a CompareValidator. You can add these rows and controls as you just did in Design view, or directly in Source view. Either way, the markup will look something

like that listed in Example 5-4. Be sure to set the correct `ControlToValidate` attributes of all these new validation controls, as well as the other attributes.

Example 5-4. *You can perform password validation using the CompareValidator control by comparing one control to another*

```
<!-- Text fields for passwords -->
<tr>
   <td>Enter your password:</td>
   <td>
      <asp:TextBox ID="txtPassword1" runat="server"
         TextMode="Password" Width="80"></asp:TextBox>
   </td>
   <td>
      <asp:RequiredFieldValidator ID="rfvTxtPassword1" runat="server"
         ControlToValidate="txtPassword1"
         ErrorMessage="Please enter your password."
         SetFocusOnError="True">*
      </asp:RequiredFieldValidator>
   </td>
</tr>
<!-- Second password for comparison -->
<tr>
   <td>Re-Enter your password:</td>
   <td>
      <asp:TextBox ID="txtPassword2" runat="server"
         TextMode="Password" Width="80"></asp:TextBox>
   </td>
   <td>
      <asp:RequiredFieldValidator ID="rfvTxtPassword2" runat="server"
         ControlToValidate="txtPassword2"
         ErrorMessage="Please re-enter your password."
         SetFocusOnError="True">*
      </asp:RequiredFieldValidator>
      <br />
      <!-- Second password must match the first -->
      <asp:CompareValidator ID="cvPasswords" runat="server"
         ControlToValidate="txtPassword2"
         ControlToCompare="txtPassword1"
         SetFocusOnError="true"
         Type="String"
         Operator="Equal"
         ErrorMessage="Passwords do not match">*
      </asp:CompareValidator>
   </td>
</tr>
```

Go ahead and test it out. If the strings you enter don't match, the control will fail validation.

The first new table row contains the `TextBox` control with its `TextMode` attribute set to `Password`. It also contains a `RequiredFieldValidator` to ensure the user doesn't leave the field blank.

The second row contains a second password text box and a second `RequiredFieldValidator` (again, the user cannot leave the field blank), but it uses a `CompareValidator` to check the value of its associated `TextBox` (`txtPassword2`) against that of the first `TextBox` (`txtPassword1`) to verify they both have the same content. The `Operator` property is set to `Equal`, and the `Type` property is set to `String`, so the two strings must match. Notice the two properties set:

```
ControlToValidate="txtPassword2"
ControlToCompare="txtPassword1"
```

Both text boxes must have a `RequiredField` validator. If the `CompareValidator` compares a string against a null or empty string value, it will pass validation.

Range Checking

At times, you'll want to validate that a user's entry falls within a specific range. That range can be within a pair of numbers, characters, or dates. In addition, you can express the boundaries for the range by using constants or by comparing its value with values found in other controls.

In this example, you'll prompt the user for a number between 10 and 20, and then validate the answer to ensure it was entered properly. To do so, create a new web site named `RangeValidator`. You'll create this exercise entirely in Design mode. To begin, drag four controls onto your page: a label, a text box, a button, and of course, a `RangeValidator` control, as shown in Figure 5-6.

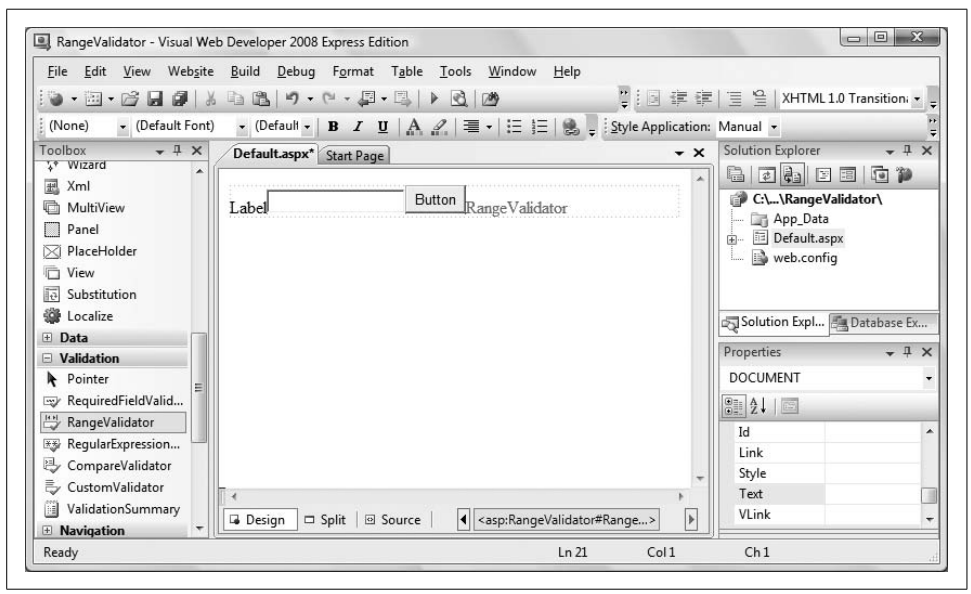


Figure 5-6. *Create the RangeValidator page in Design mode. Notice how the RangeValidator control shows up.*

Click on the Label and set its Text property to "Enter a number between 10 and 20:". Click on the TextBox, set its ID to txtValue. Click the button and set its Text to Submit. Finally, click on the RangeValidator, and in the Properties window, click Type. Choose Integer from the drop-down list. Set the MinimumValue property to 10 and the MaximumValue property to 20. Next, click on the ControlToValidate property, pick the text box, and set the Text property to "Between 10 and 20 please."

Run your application. Enter a value and click Submit. The text "Between 10 and 20 please" will be displayed if the value is not within the range of values specified by the MinimumValue and MaximumValue attributes. The Type attribute designates how the value should be evaluated and may be any of the following types: Currency, Date, Double, Integer, or String.

If there are no validation errors, the page can be submitted; otherwise, the range checking error message is displayed.

If the user leaves the field blank, the validation will pass and the page will be submitted. You'll want to ensure *some* value is entered, so add a RequiredFieldValidator in addition to the RangeValidator.

Regular Expressions

Often, a simple value or range check is insufficient; you must check that the *form* of the data entered is correct. For example, you may need to ensure that a zip code is five digits with the option to accept an extra four digits, an email address is in the form *name@place.com*, credit card information matches the right format, and so forth.

A regular expression validator allows you to verify that a text field matches a *regular expression*. Regular expressions are a language for describing and manipulating text.

For complete coverage of regular expressions, see *Mastering Regular Expressions*, by Jeffrey E. F. Friedl (O'Reilly).

A regular expression consists of two types of characters: literals and metacharacters. A literal is a character you wish to match in the target string. A metacharacter is a special character or sequence of characters that acts as a command to the regular expression parser. (The parser is the engine responsible for understanding the regular expression.) Consider this regular expression:

```
^\d{5}$
```

This will match any string that has exactly five numerals. The initial metacharacter, ^, indicates the beginning of the string. The second metacharacter, \d, indicates a digit. The third metacharacter, {5}, indicates five of the digits, and the final metacharacter, $, indicates the end of the string. Thus, this regular expression matches five digits between the beginning and end of the line and nothing else.

When you use a `RegularExpressionValidator` control with client-side validation, the regular expressions are matched using JScript, the Microsoft version of JavaScript. This may differ in small details from the regular expression checking done on the server.

A more sophisticated algorithm might accept a five-digit zip code or a nine-digit zip code in the format of 12345-1234 by using the | metacharacter, which represents the "or" operator. Rather than using the `\d` metacharacter, you could designate the range of acceptable values:

```
[0-9]{5}|[0-9]{5}-[0-9]{4}
```

To see how this works, make a copy of the `RangeValidator` web site you just created, and name it `RegularExpressionValidator`. Change the `Label` text to "Enter a U.S. zip code:". Replace the `RangeValidator` control with a `RegularExpressionValidator` control.

Use the Properties window to set the `ControlToValidate` to `txtValue` and set the text to "Please enter a valid U.S. zip code." Click on the property for Validation Expression, and click on the ellipsis. A Regular Expression Editor pops up with a few common regular expressions; you can also enter your own. Scroll down and choose U.S. ZIP code, as shown in Figure 5-7.

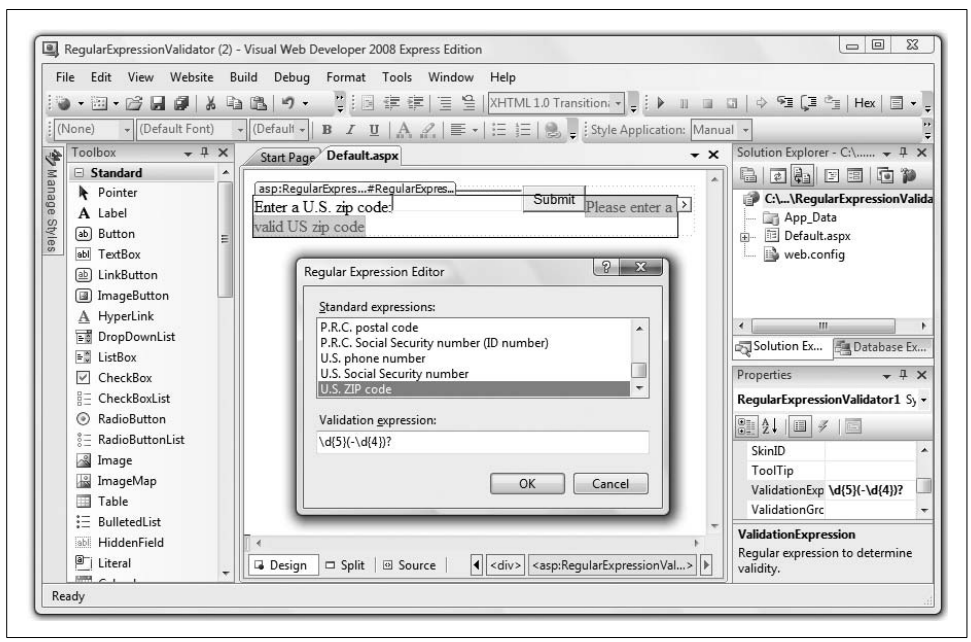


Figure 5-7. *The Regular Expression Editor makes it a snap to use the RegularExpressionValidator. Just select the ValidationExpression in the Properties window, and then click the ellipsis button to open the editor.*

Run the program, and test out the field by entering some responses. You will see that anything with a format of either a standard five digit zip code or a "Zip+4" will pass validation, but anything else will fail. Note, though, that this validator simply checks the format of the input; it doesn't check whether the input is actually a valid zip code.

If you choose "Custom," the Validation expression box will be blank, allowing you to enter any expression you choose. For help with creating custom regular expressions, we recommend the program *RegEx Buddy* (*http://www.RegExBuddy.com*).

Custom Validation

There are times when the validation of your data is so specific to your application that you will need to write your own validation method. The `CustomValidator` is designed to provide all the infrastructure support you need. You write a method that tests the user's input in whatever way you need, and returns a Boolean value: `true` or `false`. Then, you add a `CustomValidator` control to the page and point it to your validation method. The `CustomValidator` control takes care of all the rest of the work.

Because validation can be performed on the client (depending on the browser) and the server, the `CustomValidator` has attributes for specifying a server-side and client-side method for validation. The server-side method can be written in any .NET language, such as C# or VB.NET, but the client-side method must be written in a scripting language understood by the browser, such as VBScript or JavaScript.

The code functionality is duplicated on the server for two reasons. First, as mentioned at the beginning of this chapter, it prevents a malicious user from bypassing the client-side validation, and second, it makes the page compatible with older browsers that may not support client-side validation or browsers that have script disabled.

To get you started, once again copy the `RegularExpressionValidator` web site to a new site named `CustomValidator`. In this example, you want to ensure that the user enters an even number.

This time, you'll report an error if the number is not evenly divisible by 2. You can imagine, however, that you could use this technique to perform a checksum on a credit card or ISBN number or otherwise perform complex data checking.

Most of these checks can be done more easily with a Regular Expression Validator; the custom validator should be used only as a last resort.

Replace the RegularExpressionValidator with a CustomValidator. Set the ControlToValidate field to the ID of the appropriate TextBox, and make sure that EnableClientScript is set to true (the default). Update the Label text, and set the Text property to "Please enter an even number."

CustomValidators have an additional property that can save you a lot of special coding: ValidateEmptyText:

```
ValidateEmptyText=false
```

If you set this property to false (the default), the text field will be considered invalid if it is empty, avoiding the need for the RequiredFieldValidator that you needed in the previous examples.

The key to making your custom validator work is in setting the client-side validator, which you do in the ClientValidationFunction property. Set this property to ClientValidator, which is the name of a JavaScript function you are going to write momentarily. Also, click the Events lightning bolt button, and set the ServerValidate event handler to ServerValidator, a method in the code-behind you are also going to write in just a bit.

To create the JavaScript function, add the following code directly to the markup file in Source view, between the closing </head> element and the opening <body> element:

```
<script language="javascript" type="text/javascript" >
   function ClientValidator(source, args)
   {
      if (args.Value % 2 == 0)
         args.IsValid=true;
      else
         args.IsValid=false;
      return;
   }
</script>
```

IntelliSense will attempt to help by autocompleting as you type this JavaScript. Do not be alarmed when this syntax does not appear as one of the options. This code will work. Remember, however, that JavaScript *is* case-sensitive.

In this function, the args parameter is an object that you use for validation. The user's input is in the args object's value property. The value that the user has entered is passed to this function in the args parameter passed to the script by the validator. If it is an even number, the function sets the IsValid property of the args object to true. Otherwise, it sets it to false.

The standard test for determining if an integer is even or odd is to divide by 2 and check the remainder. If the remainder is 0, the integer is even. If it's 1, the integer is odd.

The operator for determining the remainder is called the *modulus*. In JavaScript (as in most programming languages), the `%` operator represents the modulus. In other words, if your integer is stored in the variable `value`, then `value % 2` is equal to 0 if `value` is even.

In VB.NET, the modulus is represented by the `Mod` operator, not the `%` operator.

You'll implement the server-side method in the code behind file, *default.aspx.vb*. Copy the highlighted code from Example 5-5 to the code skeleton for `ServerValidator` you created above.

Example 5-5. *The server-side custom validation code tests whether a number is odd or even*

```
Protected Sub ServerValidator(ByVal source As Object, _
        ByVal args As System.Web.UI.WebControls.ServerValidateEventArgs) _
        Handles cvValue.ServerValidate
   args.IsValid = False
   Dim evenNumber As Integer = Int32.Parse(args.Value)
   If evenNumber Mod 2 = 0 Then
      args.IsValid = True
   End If
End Sub
```

This method does the same thing as the client-side validator, only in VB rather than in JavaScript. There are a few things to notice about these methods. First, the value that the `CustomValidator` is examining is passed to your routine as the `Value` property of the `ServerValidateEventArgs` event argument. You convert that string to an `int` using the `Int32` object's `Parse` method, as shown.

The declaration for the `CustomValidator` in the content file sets the client-side method and the server-side method you've designated.

```
<asp:CustomValidator ID="cvValue" runat="server"
   ControlToValidate="txtValue"
   ValidateEmptyText="false"
   ClientValidationFunction="ClientValidator">
   Please enter an even number.
</asp:CustomValidator>
```

If you run this program in a current browser and enter an odd number, the page will never be posted back to the server; the JavaScript handles the validation on the browser. If you enter an even number, however, the client-side script *and* the server-side script will run (to protect against spoofing from the client).

Summary

- Users will enter improperly formatted data into your forms, but validation can allow the controls to check that data before it's accepted by your server.
- ASP.NET provides validation controls that can check for a number of common user errors.
- Current browsers can validate input on the client side, eliminating a round trip to the server.
- The `RequiredFieldValidator` simply checks that the user has made an entry in the specified control. On `TextBoxes` and `DropDownLists`, this validator can also make certain that the user has selected an item other than the initial value.
- If you set the `SetFocusOnError` property to `true`, the focus is automatically placed on the control that fails validation, making it easier for the user to find.
- You can use the `ValidationSummary` control to provide detailed feedback to the user in a single spot on your page. You can still mark the individual controls that failed validation, but you don't need to put a lengthy error message next to the control.
- With the `CompareValidator` control, you can check the user's input against a constant value, a database value, or the value of another control. You can check if the input is greater than, less than, or equal to the specified value, or you can simply check that the input is of the desired data type.
- The `RangeValidator` control checks to see if the user's input falls within an appropriate range. You can specify the maximum and minimum values of the range.
- Regular expressions are a language that uses literals and metacharacters to describe and search text strings.
- With the `RegularExpressionValidator`, you can check that the user's input meets the expected pattern for data such as a phone number, a zip code, an email address, or other variations. The Regular Expression Editor provides some common regular expressions, or you can provide your own.
- If none of the existing controls provides the validation you need, you can use a `CustomValidator` to add custom JavaScript code to evaluate the user's input. Your custom code can do anything you like, but it can only return `true` or `false`.

You've created a lot of pages so far, and most of them have had familiar elements that you see as you browse the web every day—form controls, database access, and postbacks, among others. What you have not done so far, though, is create a page that looks like something you'd see on the Web. For that, you need style, and we don't just mean good fashion sense. In the next chapter, you'll learn how to provide a uniform, professional look to all your pages, and how to include special touches—such as navigation tools—that separate a quality web site from just a collection of controls.

Quiz

1. What is the reason for validation?
2. What do you do if you want a button to post the page without checking validation?
3. What is the best type of validator to use for a radio button list?
4. What's the difference between the `Static` and `Dynamic` values of the `Display` property?
5. Suppose the first item in your drop-down list is "Choose a payment method." How do you make sure users choose one?
6. What's the benefit of using the `ValidationSummary` control?
7. What control should you use to make sure the user can't order more of a single item than you actually have in stock?
8. Suppose you run a hotel that requires at least two guests stay in a double room, but no more than five guests. What control should you use on the "Number of guests" field?
9. How do you check that the user has entered a valid email address?
10. Suppose your theme park offers discounts to customers between the ages of 6 and 12, and also to customers over 65. What kind of control would you use to validate whether the customer is eligible for a discount, while still using a single age field?

Exercises

Exercise 5-1. In the exercises in this chapter, you're going to create a form that users can fill out if they want to participate in a friendly phone survey (I'm told some people like to get survey calls in the middle of dinner). To begin, create a page with a table with three columns, one each for a label, control, and validator. Then, add text boxes for the user's name, address, city, state, and zip code. Be sure to add the appropriate validators for each field—don't worry about the format of the input right now; you just want to make sure that something is filled in. Finally, add a Submit button. It doesn't matter too much what this form looks like, but it could look something like Figure 5-8.

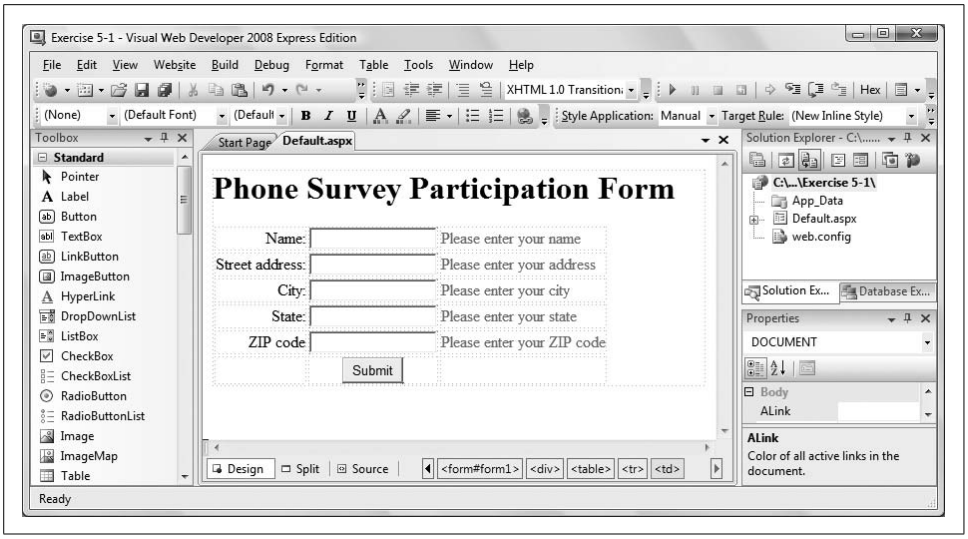


Figure 5-8. *Your goal for Exercise 5-1.*

Exercise 5-2. Let's make things a little more interesting this time. For starters, move the text from the individual validators to error messages, and add a summary control at the bottom of the form, above the Submit button. Next, you don't want anyone participating in the survey if they're under 18, so add another label and field asking for the user's age. Add appropriate validators to make sure the user isn't too young. Because you're polite, you'll ask for a date when you should call the user, but your survey is only going on in July 2009. Add another table row with a label and a field asking for a date, and add the appropriate validators to make sure the date is sometime in July 2009. Your form should look something like Figure 5-9.

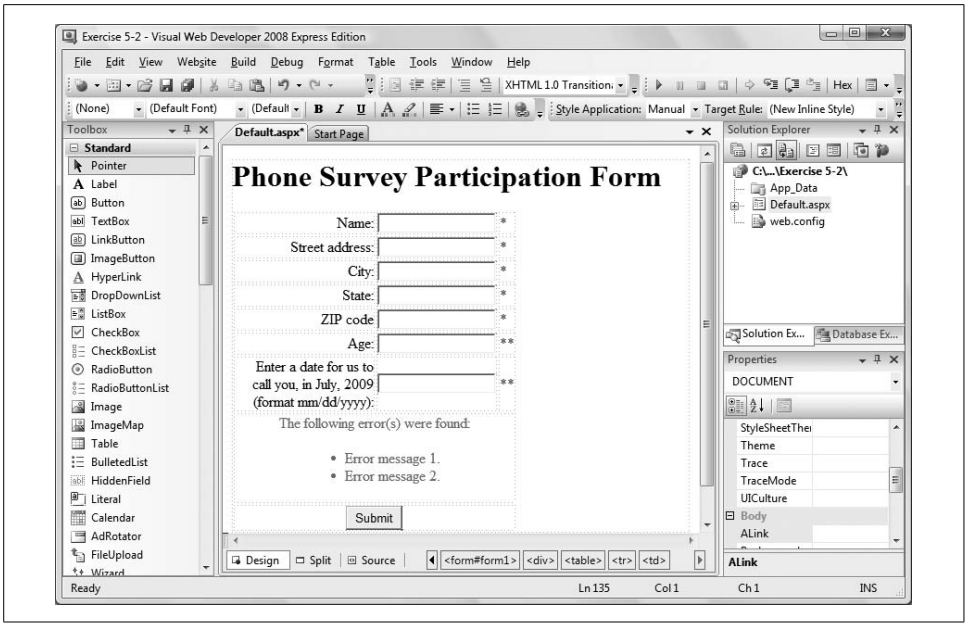


Figure 5-9. *Your goal for Exercise 5-2.*

Exercise 5-3. If the user doesn't mind being called at home, you might as well make a follow-up call to ask additional survey questions. This call still has to take place in July, but if it's a follow-up call, it would have to be *later* than the first call. Add a row to the table with a label and text box where users can enter a date for the follow-up call, and add appropriate validators to make sure the follow-up call comes after the initial call, but still in July 2009. The result should look something like Figure 5-10.

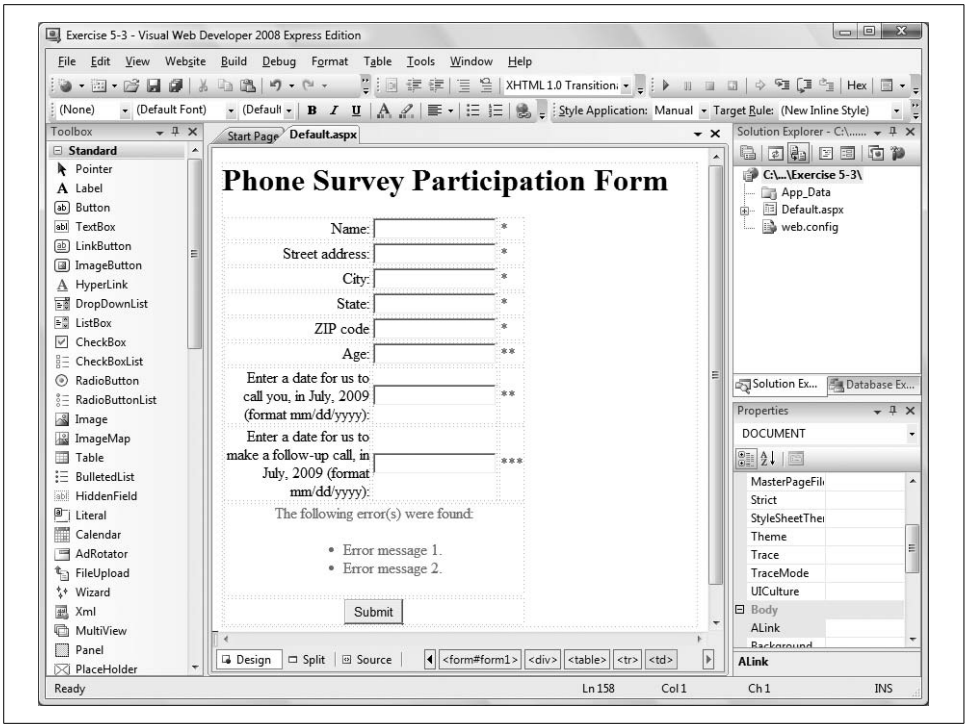


Figure 5-10. *Your goal for Exercise 5-3.*

Exercise 5-4. If you're going to call the user at home, you'll need a phone number to call. If the user is willing to give out his phone number, you might as well get his email address as well. After all, you never know when you'll need it. But if the user forgets a digit, or leaves off the ".com" from his email address, it'll do you no good. Add two more rows to the table, with labels and text fields where the user can enter a phone number and email address. Then, add the appropriate validators to make sure that the input is in the correct form. The form should look something like Figure 5-11.

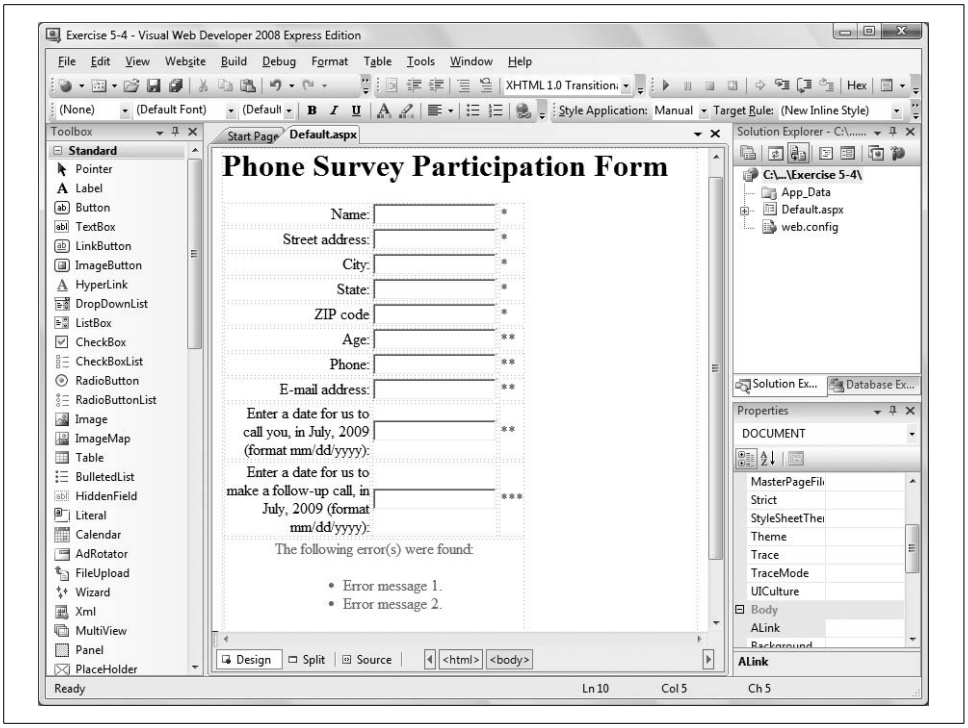


Figure 5-11. *Your goal for Exercise 5-4.*

CHAPTER 6

Style Sheets, Master Pages, and Navigation

Back in the early mists of time, when the Earth was young and the Web was new (circa 1994), we created web pages in HTML (HyperText Markup Language). After many eons (or so it seemed), we were able to add styles to the HTML elements, allowing us to take greater control over the presentation of web pages.

Eventually content (the HTML) was divided from presentation and layout through the use of *styles*, and that was good. In fact, it came to pass that presentation information was given its own file—a *style sheet*—to allow for reuse, a consistent presentation across many pages, and easier maintenance, and that was very good indeed.

Styles and style sheets are a significant (but often overlooked) tool for web developers, too often ignored by "programmers" who disparage style sheets as being in the realm of "designers"—leading to the creation of web applications that are ugly and terribly difficult to maintain.

A powerful technique for creating sites with a common look and feel across all of the pages is *master pages*, covered later in this chapter. Master pages cannot only define the layout of the pages in your site, but they can also contain menus and other navigational aids such as site maps and bread crumbs, and these, too, will be covered in this chapter.

Styles and Style Sheets

A *style* specifies how an object will be displayed on an output device, typically a browser. Styles can be used to manipulate the layout and appearance of controls and text, detailing every aspect from border color to font size to position on the page.

Web applications use styles to ensure attractive and reasonable display on a wide variety of devices, including desktop and laptop computers, tablet PCs, mobile PCs, telephones, PDAs, televisions, printers, audio devices, and media not yet imagined.

HTML and ASP.NET controls apply styles through the use of properties and attributes. There are three ways to apply styles to an element on a web page:

Inline
: The style is implemented as an attribute of a specific element.

Document
: A set of styles are defined on and for a single HTML page.

External
: A style sheet is created as a separate file and "included" in one or more HTML pages.

Cascading Styles

Style rules cascade down from the most general (the external style sheet), to the more specific (document level styles), to the most specific (styles applied to particular elements), hence the term *cascading style sheets* (CSS).

If your style sheet says that text boxes should have a white background, but one particular page says that its textboxes will have gray backgrounds, and on that page the seventh text box has its own style calling for a yellow background, the rules will cascade—style sheet, to document, to element. All other pages in your web site will have text boxes whose background color is controlled by the style sheet. Your one document will have text boxes with gray backgrounds, except for the seventh text box, which will have…you guessed it! A yellow background.

For a complete discussion of CSS, see the following books: *HTML & XHTML: The Definitive Guide*, by Chuck Musciano and Bill Kennedy, or *CSS: The Definitive Guide*, by Eric A. Meyer (both published by O'Reilly).

Inline Styles

You can apply styles to a specific element using the inline `style` attribute, as shown in the following snippet of HTML:

```
<input type="text" value="Sample text" style="color:Red;font-family:Arial;
   font-weight:bold;width:150px;" />
```

The `style` attribute contains one or more style properties, each consisting of a property name and value separated by a colon. Each property-value pair is separated from the next pair by a semicolon.

When you're using ASP.NET controls, you may set inline styles either in the markup or as properties in design view.

To see this, create a new web site called `AspNetInLineStyles`. Switch to Design view and drag a `TextBox` control from the Standard section of the Toolbox onto the page.

In the Properties window, set the following properties (you'll need to expand the Font group to set the first two properties).

Property	Value
Font-Bold	True
Font-Name	Arial
ForeColor	Red
Text	Sample Text
Width	150px

The resulting Design view should look something like Figure 6-1.

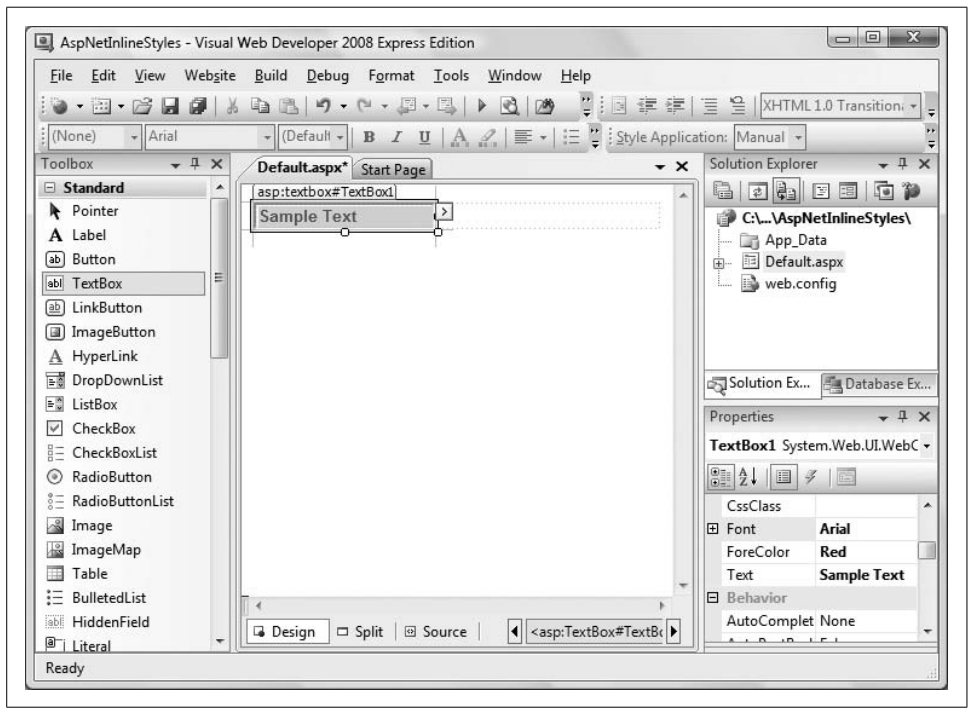


Figure 6-1. *You've set the inline style properties on this TextBox by using the IDE.*

When you set the Font Name property, the IDE automatically fills in the Names property for you.

Run the application. When the page comes up in the browser, view the source by clicking on the View → Source menu item in IE6 or Page → View Source in IE7.

Notice how this ASP.NET TextBox is rendered to the page:

```
<input name="TextBox1" type="text" value="Sample text" id="TextBox1"
    style="color:Red;font-family:Arial;font-weight:bold;width:150px;" />
```

It is the same as if you had coded HTML with inline styles, which is in fact what ASP.NET sends to the browser based on the ASP.NET controls and their properties.

You can also set or change style properties programmatically (as you can any control property).

To see this, close the browser, then drag a Button control from the Standard section of the Toolbox onto the page. Using an If-Then-Else statement, change its Text property to toggle between two colors.

Double-click the Button in Design view to open up the event handler for the Click event. Enter the highlighted code from Example 6-1.

Example 6-1. *The ButtonClick Event handler toggles the inline styles*

```
Protected Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
    If TextBox1.ForeColor = Drawing.Color.Red Then
        TextBox1.ForeColor = Drawing.Color.Green
    Else
        TextBox1.ForeColor = Drawing.Color.Red
    End If
End Sub
```

Run the application. Each time you click the button, the ForeColor property will toggle between Red and Green, which it does by changing the style attribute of the HTML that the page renders.

Pros and cons

Inline properties are excellent for overriding the styles in your existing stylesheet for a particular control. Unfortunately, they are very easy to use instead of style sheets, and programmers are often seduced into using inline styles to excess, creating markup that is very difficult to maintain.

Document-Level Styles

Just as you can use inline styles to override a style sheet for a single control, you can also add styles to a single document to set or override a particular setting for that one page. This is an error-prone technique for any multipage web site (that is, for virtually any serious web site), so we will be brief in our presentation of how to use them.

Document level styles are added to a page with a <style> element in the <head> section of the page as shown in Example 6-2. (You don't need to create this one yourself.) In this web site, called *AspNetDocumentLevelStyles*, the style for the top-level

heading, <h1>, will be overridden to display in red, bold, and italicized text. Also, a new style will be defined, called GreenText.

Example 6-2. *Default.aspx for AspNetDocumentLevelStyles; notice the document styles defined within the comment tags*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
      Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>ASP.NET Document Level Styles</title>
    <style type="text/css">
      <!--
         h1 {color:Red; font-style:italic; font-weight:bold;}
         .GreenText {color:Green;}
      -->
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>
      <h1>Custom H1 Heading</h1>
      <asp:Label ID="Label1" runat="server"
         Text="Green text in an ASP.NET Label control."
         CssClass="GreenText"></asp:Label>

    </div>
    </form>
</body>
</html>
```

Note the period in front of the GreenText style name; it is required. Also, style names are case-sensitive.

Also, notice that the style definitions are embedded between HTML comment characters. This is for the benefit of very old browsers that may not recognize styles; they will ignore the styles enclosed in comments.

Figure 6-2 displays the results of these style settings.

Pros and cons

It is tempting to use a document-level style either to set the styles for that page or to override the general styles for the entire site. This can be effective, but tends to be hard to maintain.

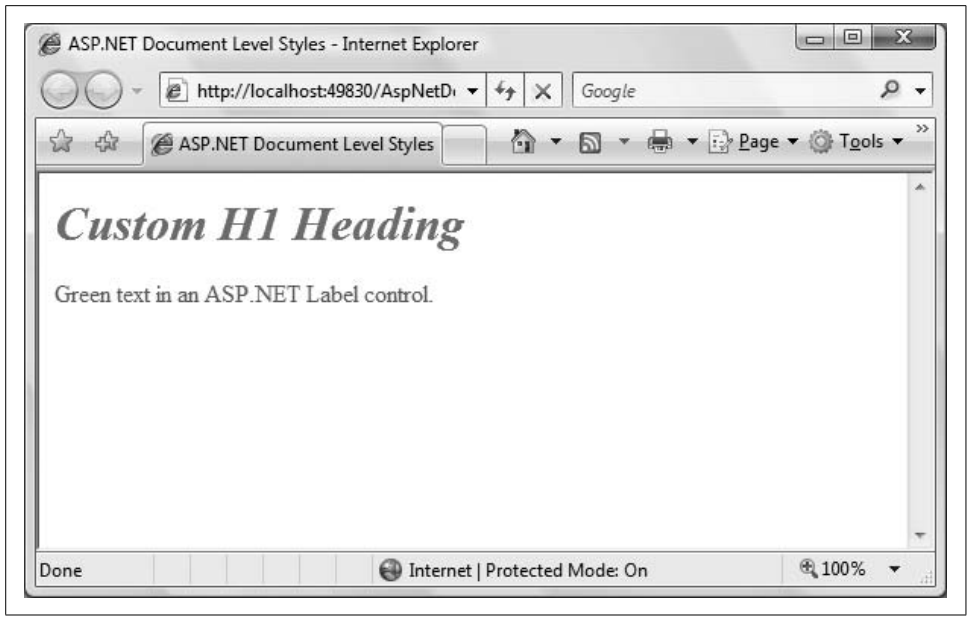


Figure 6-2. *The result of using document-level styles. It looks perfectly fine for this one page, but across a whole site, using document-level styles is tough to maintain.*

Experience shows that collecting styles into a set of external style sheets, even if some styles are targeted at a subset of pages (where that subset could be as small as a single page) tends to be far easier to maintain in the long term.

External Style Sheets

The net result is that in most applications, styles are defined in an external file, called (somewhat confusingly) a *style sheet* (or, as already mentioned, a "Cascading Style Sheet," or even an "External Style Sheet"). This style sheet is "linked" or "imported" into each page by a directive at the top of the *.aspx* page.

To complicate things further, you are not limited to a single style sheet for your entire application. In fact, you are free to create separate style sheets for different sections of your application, or for rendering to different devices.

You first saw the use of a CSS style sheet back in Chapter 3, where you used styles to differentiate the watermarked and unwatermarked text boxes. Look at the CSS Style sheet created for that application, reproduced here in Example 6-3.

Example 6-3. *The StyleSheet.css for the AdventureWorksWatermarks web site is very simple, but style sheets can be much longer and more complex*

```
body {
}
.watermarked {
   padding:2px 0 0 2px;
```

Example 6-3. *The StyleSheet.css for the AdventureWorksWatermarks web site is very simple, but style sheets can be much longer and more complex (continued)*

```
   border:1px solid #BEBEBE;
   background-color:#F0F8FF;
   color:gray;
   font-family:Verdana;
   font-weight:lighter;
}
.unwatermarked {
   height:18px;
   width:148px;
   font-weight:bold;
}
```

There are two *style classes* in this style sheet: `watermarked` and `unwatermarked`.

Style classes are defined with a leading period, followed by the name of the class, and then the definition of the style class enclosed in braces. To use a style class, the element must specifically identify the class it wants to use, such as:

```
<asp:TextBox CssClass="watermarked"...
```

When you're applying a style to a web server control, you use the `CssClass` property. When you're applying a style to an HTML control, you use the `class` property instead.

Your style sheet can also define styles for "selectors," and these styles will automatically be applied to matching elements. For example, if you define a selector for the `<p>` (paragraph) element, all paragraph elements would have that style applied.

The application of styles to both classes and selectors can become complex once you begin nesting, which is why we strongly recommend reading a solid book on CSS syntax (such as those mentioned in the note above).

The names used for style rules and classes are case sensitive. If your style sheet has a class called `watermarked` and you assign the class name `Watermarked` (with a capital W) it will be ignored, with no error message, leading to many happy hours of debugging.

To see style sheets at work, create a new web site called *AspNetExternalStyles*.

Click on the Website → Add New Item… menu item, and select Style Sheet from the list of templates. You could use the default name of *StyleSheet.css*, as we will only be using a single style sheet in this web site, but let's call it *MyStyleSheet.css*.

The new style sheet will be visible on the editing surface, with an empty *selector* for the body element:

```
body {
}
```

You will now add additional style classes for three different headings and the body text, plus an overridden selector for all paragraph elements.

Right-click anywhere on the editing surface and select Add Style Rule… to bring up the Add Style Rule dialog box. Select the Class Name radio button and enter the style class name `.MyHeading1` (either with or without the leading period—the style will be inserted into the style sheet with exactly a single leading period in either case). Click OK; this will insert an empty style class called MyHeading1, so your style sheet will now look like the following:

```
body {
}
.MyHeading1
{
}
```

Place the mouse pointer anywhere between the curly braces following `MyHeading1`, right click, and select Build Style to bring up the Modify Style dialog box.

Add the styles listed in Table 6-1.

Table 6-1. *Styles in .MyHeading1*

Category	Style	Value
Font	font-family	Arial, Helvetica, sans-serif
	font-size	xx-large
	font-weight	bold
	font-style	normal
	Color	#FF0000 (Red)
Block	line-height	1.25 in
Background	background-color	#00FFFF (Aqua)
Box	padding-top	.75 in
(Uncheck "Same for all")	padding-right	.5 in
	padding-bottom	0.25 in
	padding-left	.5 in

When you are done, it should look like Figure 6-3 (showing the Font styles).

Click OK to accept the style modifications, and you will see the style attributes added to the `MyHeading1` class in *MyStyleSheet.css*.

That's just one of the heading styles, however; you still have two others to go, plus a body text style. You can either add the additional styles in a similar manner, using the values in Example 6-4, or you can edit the style sheet directly. In any event, create the styles so that the finished style sheet looks something like Example 6-4.

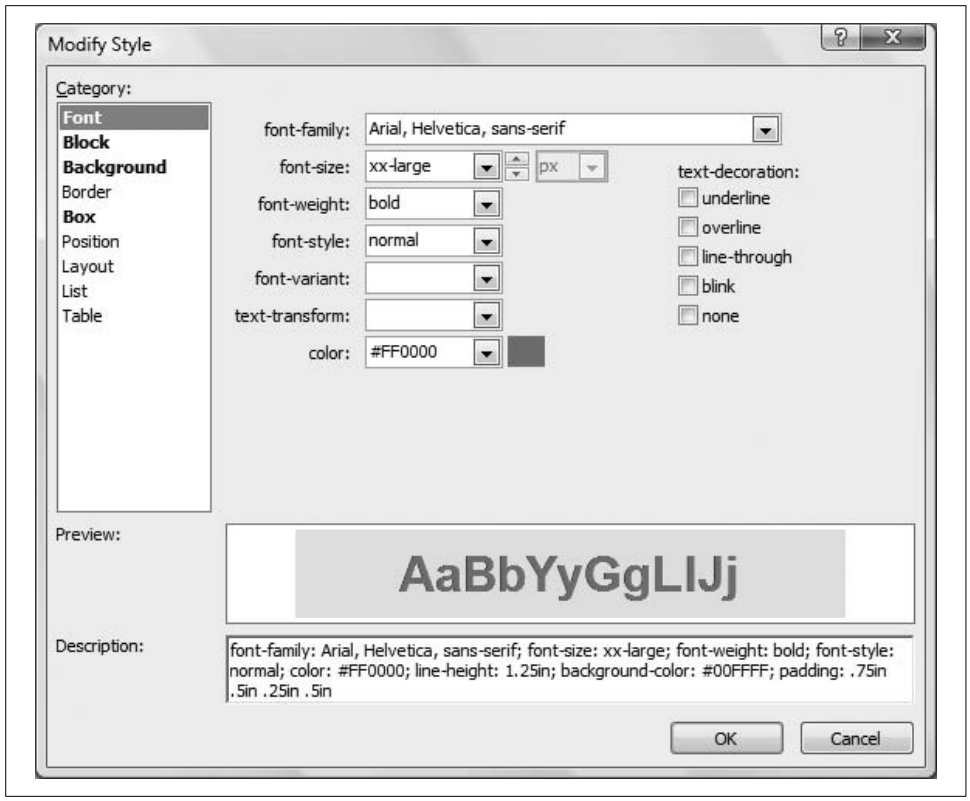


Figure 6-3. *There are a lot of different style attributes for the .MyHeading1 style class, but the Modify Style dialog makes them easy to keep track of.*

As you type, IntelliSense will show all possible style attributes and provide hints for valid values.

Example 6-4. *You can edit MyStyleSheet.css directly if you like, or you can use the Modify Style dialog to do it for you*

```
body {
}
.MyHeading1
{
   padding: .75in .5in .25in .5in;
   font-family: Arial, Helvetica, sans-serif;
   font-size: xx-large;
   font-weight: bold;
   font-style: normal;
   color: #FF0000;
   background-color: #00FFFF;
   line-height: 1.25in;}
.MyHeading2
{
   font-family: Arial, Helvetica, sans-serif;
```

Example 6-4. *You can edit MyStyleSheet.css directly if you like, or you can use the Modify Style dialog to do it for you (continued)*

```
   font-style:normal;
   font-weight:bold;
   font-size:x-large;
   background-color:Aqua;
   color:Blue;
   padding-left:.1in;
   padding-right:.1in;
   padding-top:.2in;
   padding-bottom:.2in;
   line-height:.75in;
}
.MyHeading3
{
   font-family: Arial, Helvetica, sans-serif;
   font-style:normal;
   font-weight:bold;
   font-size:large;
   color:Black;
}
.BodyText
{
   font-family:Times New Roman, Serif;
   font-style:normal;
   font-weight:bolder;
   font-size:medium;
}
p
{
   color:Blue;
}
```

You can specify colors either using named colors, e.g., Red, Black, or Aqua, or using RGB values, e.g., #FF0000. The IDE tools insert RGB values, but when you're editing manually, it is often easier to use named colors unless you must specifically match a very precise color. RGB specifies colors by relative amounts of red, green, and blue. The value always begins with the pound sign, then three pairs of characters, each of which is a hexadecimal number from 00 through FF, representing red, green, and blue, respectively.

There are a number of possible units of measurement you can use for attributes that require absolute values, such as padding and line-height. In addition, *Relative* units, which are relative to the other content on the page, are also available.

Before the styles in a style sheet will take effect, you need to attach the style sheet to the page. You can do this one of two ways, which have essentially the same result. The first way is to manually enter an `@import` statement inside a `style` element between the opening and closing `head` tags in the markup file:

```
<head runat="server">
    <style type="text/css">@import url(MyStyleSheet.css);</style>
```

```
    <title>Untitled Page</title>
</head>
```

In this case, a relative URL is provided, which refers to your style sheet in the current directory. Because it is a URL, it can be either relative or absolute. For example, you could provide an absolute URL such as *http://CorporateWebSite.com/stylesheets/handhelds.css*.

The `@import` command must appear in the `<head>` element, and before any conventional style rules are specified. Otherwise, the imported style sheet will be ignored. This allows the browser to properly cascade styles from the external style sheet down to the element-level styles.

The second technique is to use a `link` tag, which also goes inside the `head` tags in the markup file. You can also enter this manually, but the IDE provides an easier way. Make sure that the *.aspx* page is visible in Design view. Click on the View → Manage Styles menu item to get the Manage Styles window shown in Figure 6-4.

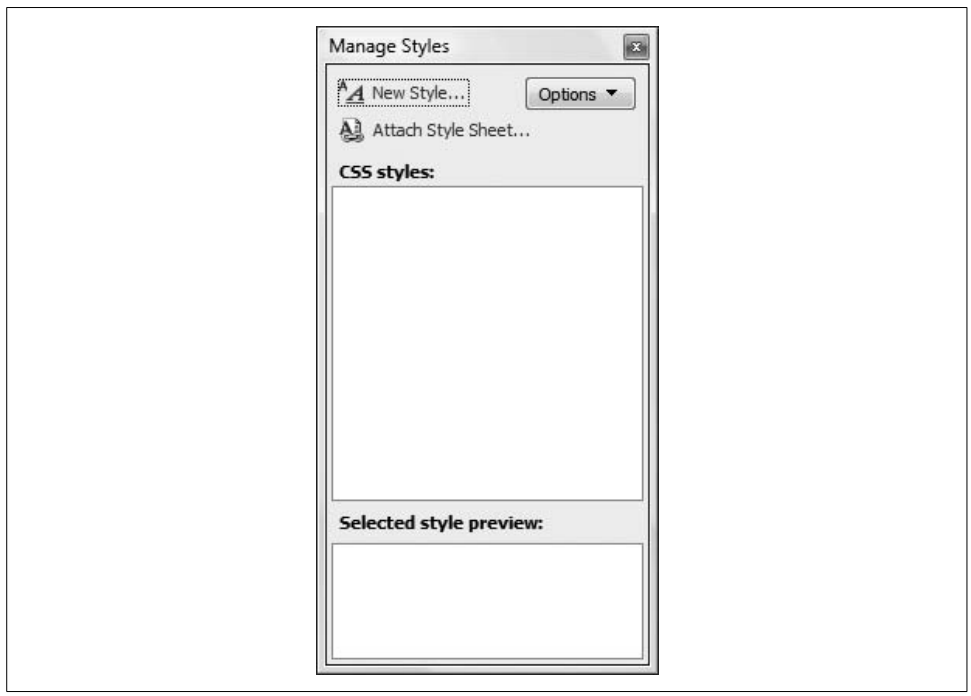


Figure 6-4. *Clicking on View → Manage Styles brings up this window for attaching a style sheet to the page, then working with that style sheet.*

There should be three styles-related items under the View menu item: CSS Properties, Manage Styles, and Apply Styles. If you don't see these items while looking at the page, switch to Design view. Then they should be visible in either Design, Split, or Source view.

Click on the Attach Style Sheet… link. This will bring up a Select Style Sheet dialog box, listing all the available style sheets. In this case, there will be only a single style sheet available: *MyStyleSheet.css*. Select it and click OK. This will automatically add the following highlighted line to the *.aspx* file:

```
<head runat="server">
   <title>Untitled Page</title>
   <link href="MyStyleSheet.css" rel="stylesheet" type="text/css" />
</head>
```

Now that you have created the style sheet with several styles and attached the style sheet to the page, you will add controls to the page and apply the styles. Open the *default.aspx* file in the IDE. Switch to Design view, and drag a `Label` control onto the page. Set the `Text` property to "Heading 1". Set the `CssClass` property to `MyHeading1` by clicking on `CssClass` in the Properties window and selecting from the available styles that you created earlier.

Type in some text on the page, select it, and apply the paragraph style to it by selecting "Paragraph <p>" from the Block Format drop-down in the Style Application toolbar (indicated in Figure 6-5).

Now, drag a second `Label` control onto the page. Set its `Text` property to "Heading 2" and the `CssClass` property to `MyHeading2`. Type some more text on the page after the label.

Drag a third label on the page. Set the `Text` property to "Heading 3" and the `CssClass` property to `MyHeading3`. Type some more text on the page.

Add two more `Label` controls with the `CssClass` set to `MyHeading2` and `MyHeading3`, along with some text.

To format the text under both `MyHeading3` headings, switch to Source view and insert some `<div>` or `<span>` elements so that you can apply the `BodyText` style, like this:

```
<span class="BodyText">Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut dolore magna aliqua.</span>
```

To apply a style to an ASP.NET control, use the `CssClass` property. However, to apply a style to an HTML control, use the `class` property.

And, just as a bit of HTML refresher, both `<div>` and `<span>` elements are used primarily to apply style to some content. The difference is that a `<div>` element incorporates a line break before and after, while a `<span>` element does not, displaying its content inline with its container. It is convenient to think of a `<div>` as creating a block, while a `<span>` delineates a series of inline characters.

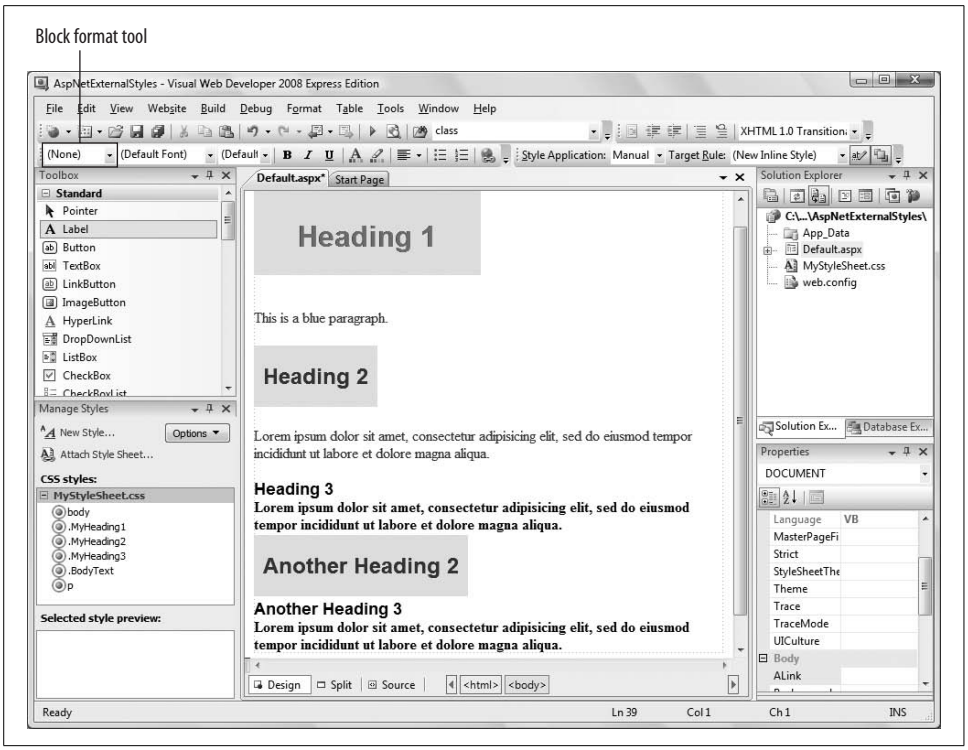


Figure 6-5. *Your AspNetExternalStyles page, in Design view, where you can see all the styles applied.*

The Design view will now look something like Figure 6-5.

Example 6-5 presents the Source view for *Default.aspx*.

Example 6-5. *The Default.aspx for AspNetExternalStyles shows how the style classes are applied to each element, but the style definitions aren't in this file*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
      Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Untitled Page</title>
   <link href="MyStyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
   <form id="form1" runat="server">
   <asp:Label ID="Label1" runat="server" CssClass="MyHeading1" Text="Heading 1">
   </asp:Label>
   <p>
      This is a blue paragraph.</p>
```

Example 6-5. *The Default.aspx for AspNetExternalStyles shows how the style classes are applied to each element, but the style definitions aren't in this file (continued)*

```
   <asp:Label ID="Label2" runat="server" CssClass="MyHeading2" Text="Heading 2">
   </asp:Label>
   <p>
      Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
      incididunt ut labore et dolore magna aliqua.</p>
   <asp:Label ID="Label3" runat="server" CssClass="MyHeading3" Text="Heading 3">
   </asp:Label>
   <div class="BodyText">
      Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
      incididunt ut labore et dolore magna aliqua.
   </div>
   <asp:Label ID="Label4" runat="server" CssClass="MyHeading2"
      Text="Another Heading 2">
   </asp:Label>
   <br />
   <asp:Label ID="Label5" runat="server" CssClass="MyHeading3"
      Text="Another Heading 3">
   </asp:Label>
   <div class="BodyText">
      Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor
      incididunt ut labore et dolore magna aliqua.
   </div>
   </form>
</body>
</html>
```

The resulting page is shown in Figure 6-6.

Master Pages

A *master page* acts as a shell or frame shared by all the pages (or some of the pages) on your site. It is common to put a logo and perhaps a menu into the master page so that these elements appear at the same location on every page without your having to recode them.

Within the master page, you place one or more content placeholder areas, which will be filled with the contents of each of the content pages. This is shown in Figure 6-7.

Unlike cascading style sheets (CSS), which help ensure that similar controls have similar appearances (see the previous section), master pages ensure that all the pages on your site have common elements such as logos, headings, footers, or navigation aids.

To use master pages, follow these steps:

1. Create a new web site.
2. Add a master page to the site.
3. Add content pages that fit the placeholders on the master page.

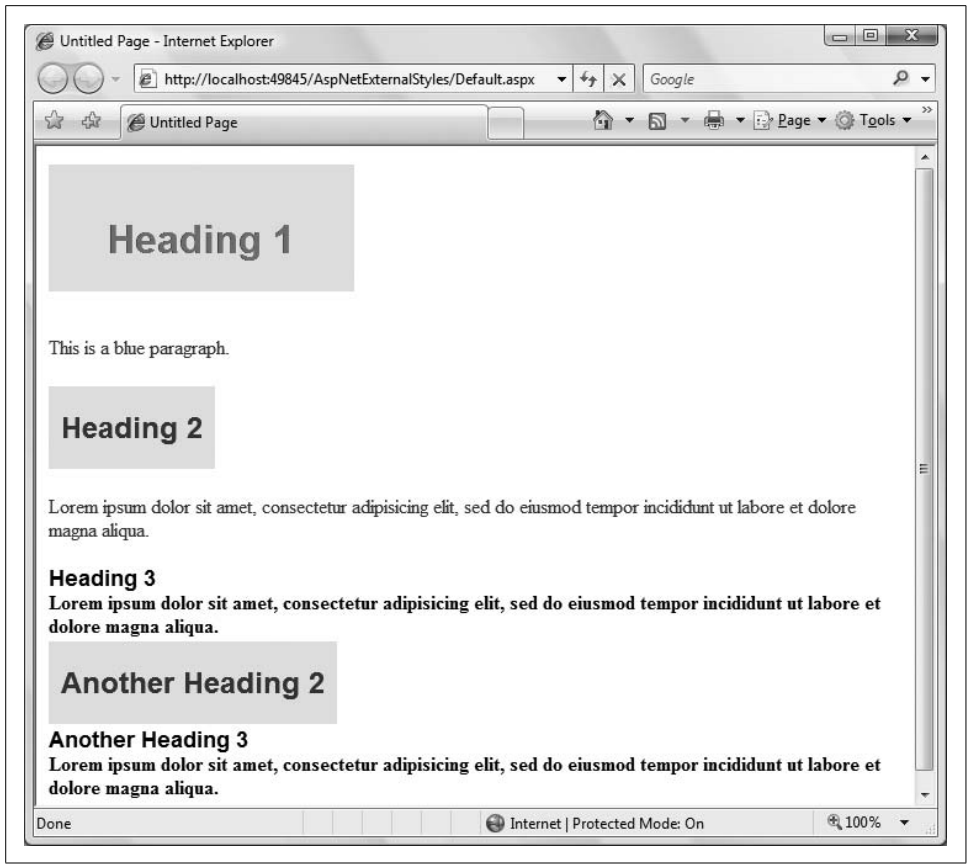

Untitled Page - Internet Explorer
http://localhost:49845/AspNetExternalStyles/Default.aspx
Google
Untitled Page
Page
Tools
Heading 1
This is a blue paragraph.
Heading 2
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Heading 3
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Another Heading 2
Another Heading 3
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Done
Internet | Protected Mode: On
100%

Figure 6-6. *This is what the AspNetExternalStyles page looks like in your browser.*

Creating a Master Page

To begin, create a new web site and call it *MasterPages*. Once the new site opens click on Website → Add New Item…. Select Master Page in the dialog box and give it the name, *CorpMasterPage.master*, as shown in Figure 6-8. Be sure to check the "Place code in separate file" checkbox, as indicated in the figure.

Even though we don't actually use any server-side code in the master page in this chapter, it is good practice to segregate all your server-side code in a code-behind file, rather than in a script block in the markup file.

Your new master page has been created with two `ContentPlaceHolder` controls already in place, one inside the `head` tags, and one in the body of the page.

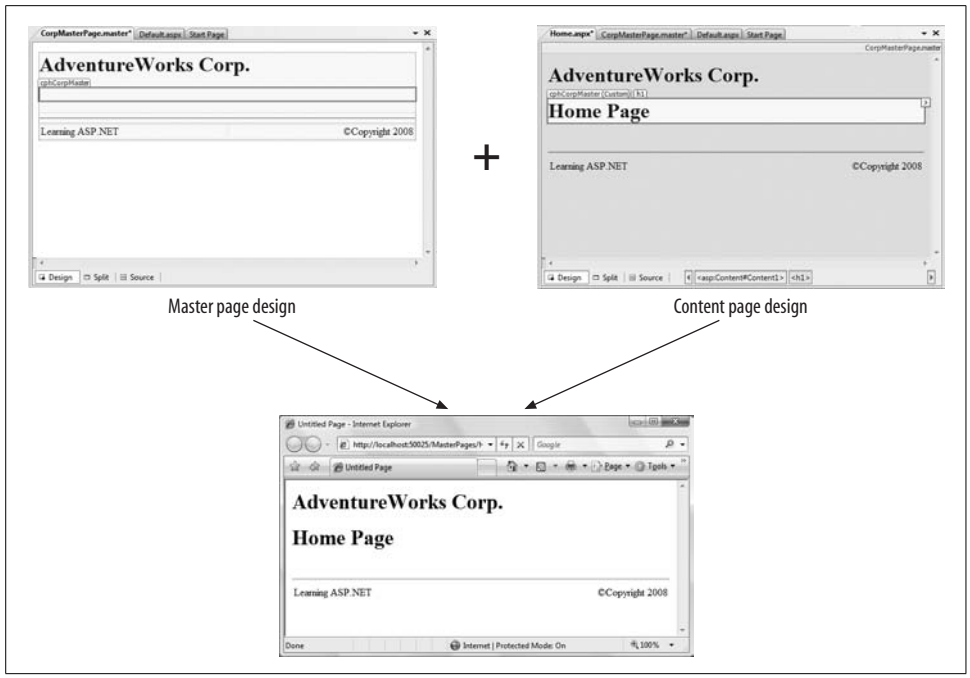


Figure 6-7. *The master page defines the content that should appear on every page of your site, and has placeholders for the content of the individual content pages. When you put them together, you get a web site with a uniform appearance.*

All master pages *must* have the extension *.master*.

Switch to Source view and change the `ID` of the placeholder in the body of the page to something more meaningful, such as `cphCorpMaster`, as in the following code:

```
<asp:ContentPlaceHolder ID="cphCorpMaster" runat="server">
</asp:ContentPlaceHolder>
```

You can have more than one `ContentPlaceHolder` control on a master page (each has its own ID). This gives you tremendous flexibility in laying out your pages, though experience shows that the majority of sites actually use only a single `contentPlaceHolder` per master page. To keep this example simple, delete the `<asp:ContentPlaceHolder>` that is within the head tags.

Normally, content inside the `head` tags is not displayed directly by the browser, but is used to convey information about the page, such as the title, styles, metadata, etc. You can also include certain ASP.NET controls, such as the `ContentPlaceHolder` and `Label`, which will display at the top of every page.

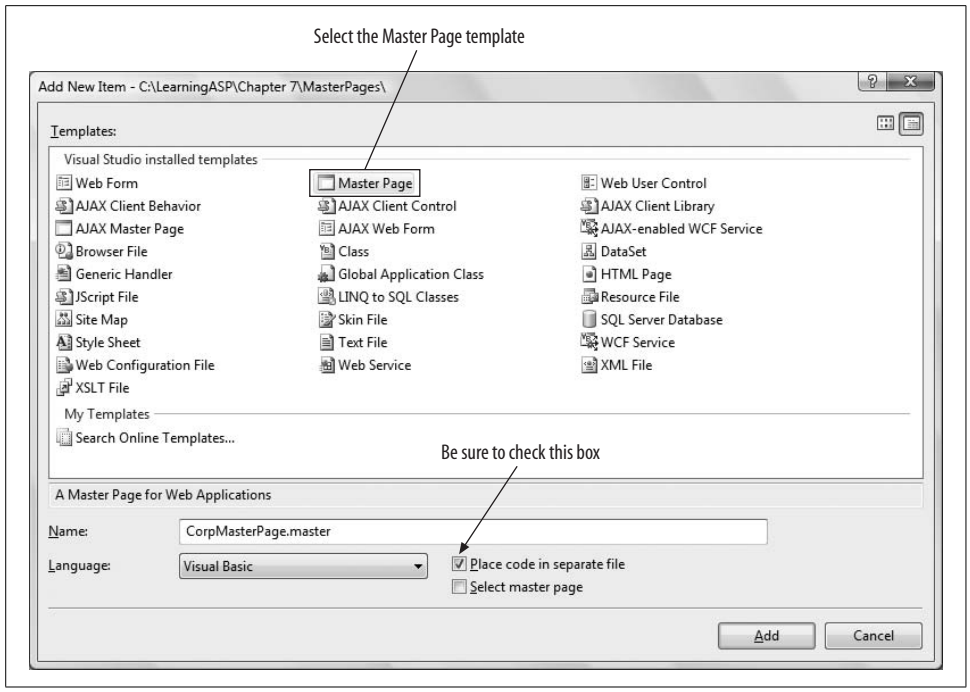


Figure 6-8. *You add a master page from the Website → Add New Item dialog. Be sure to check the "Place code in separate file" checkbox.*

The placeholder will be filled by the contents of the content page, which in turn will be surrounded by whatever else you place on the master page. Within the master page, you may add anything you like surrounding the `ContentPlaceHolder` control. For example, you might add a logo at the top of the page and a copyright notice at the bottom. Perhaps you may want navigation controls to appear consistently positioned along the side of your pages. You can even add other content placeholders, giving each a unique ID.

For this example, place an `<h1>` header on the page above the `ContentPlaceHolder` within the body of the page, and an HTML table below as a footer. The Source view should look something like Example 6-6. Add the highlighted code to your page.

Example 6-6. *You'll be adding some basic content to CorpMasterPage.master*

```
<%@ Master Language="VB" CodeFile="CorpMasterPage.master.vb"
      Inherits="CorpMasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Untitled Page</title>
</head>
```

Example 6-6. *You'll be adding some basic content to CorpMasterPage.master (continued)*

```
<body>
   <form id="form1" runat="server">
   <div>
      <h1>AdventureWorks Corp.</h1>
      <asp:ContentPlaceHolder ID="cphCorpMaster" runat="server">
      </asp:ContentPlaceHolder>
      <br />
      <hr />
      <table width="100%">
         <tr>
            <td width="50%" align="left">Learning ASP.NET</td>
            <td width="50%" align="right">&copy;Copyright 2008</td>
         </tr>
      </table>
   </div>
   </form>
</body>
</html>
```

Switch to Design view; the master page will look something like that shown in Figure 6-9.

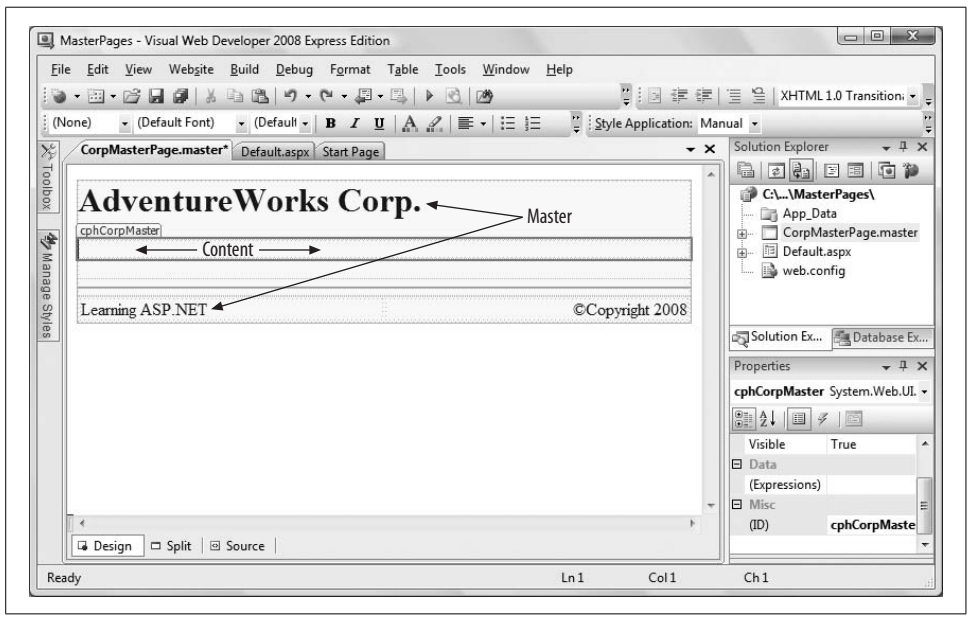


Figure 6-9. *This is what your master page looks like in Design view. The heading at the top and the footer at the bottom will be applied to all of your content pages. The content from your content pages will appear in the placeholder, which doesn't look very large when it's empty, but it expands as you add things.*

Adding Content Pages

The pages you'll add that will use this master page will put all of their content into the `ContentPlaceHolder` defined in body of the master page. When combined, the two create a content page.

For this example, you'll add two new *.aspx* pages, *Home.aspx* and *SecondPage.aspx*. There are two ways to do this.

One way is to click on the Website → Add Content Page menu item. However, this will produce a page with a default name, which you will almost certainly want to change, which is more of a nuisance than it is worth.

The better way is to click on Website → Add New Item…, then add a "normal" Web Form. Call the new page *Home.aspx*, and be sure to check both the "Select master page" and "Place code in separate file" checkboxes, as indicated in Figure 6-10.

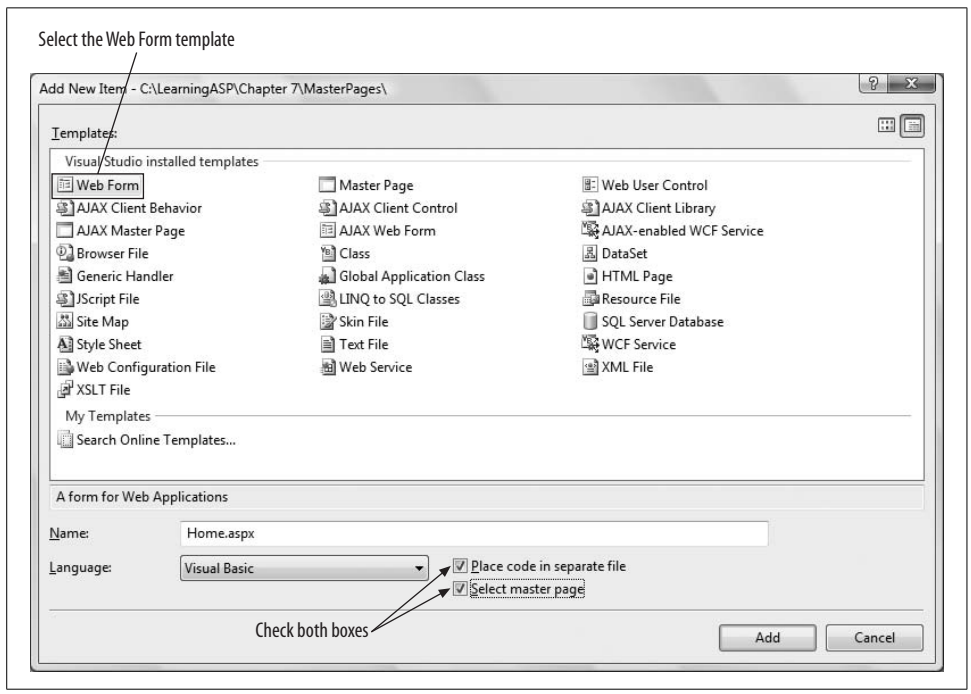


Figure 6-10. *The easiest way to add a content page is by adding a new Web Form, and making sure to check the "Place code in separate file" and "Select master page" checkboxes.*

When you click the Add button, the Select a master page dialog will open. Choose *CorpMasterPage.master* (the only master page available at this point), and click OK.

Then, switch to Design view.

Your new *Home.aspx* page will be shown within the master page. The Content box will allow you to add any content you like, including controls, text, and so forth. The contents of the master page will be visible, but inaccessible.

Add some text and format it as HTML Heading 1 using the Block Format drop-down menu, as indicated in Figure 6-11.

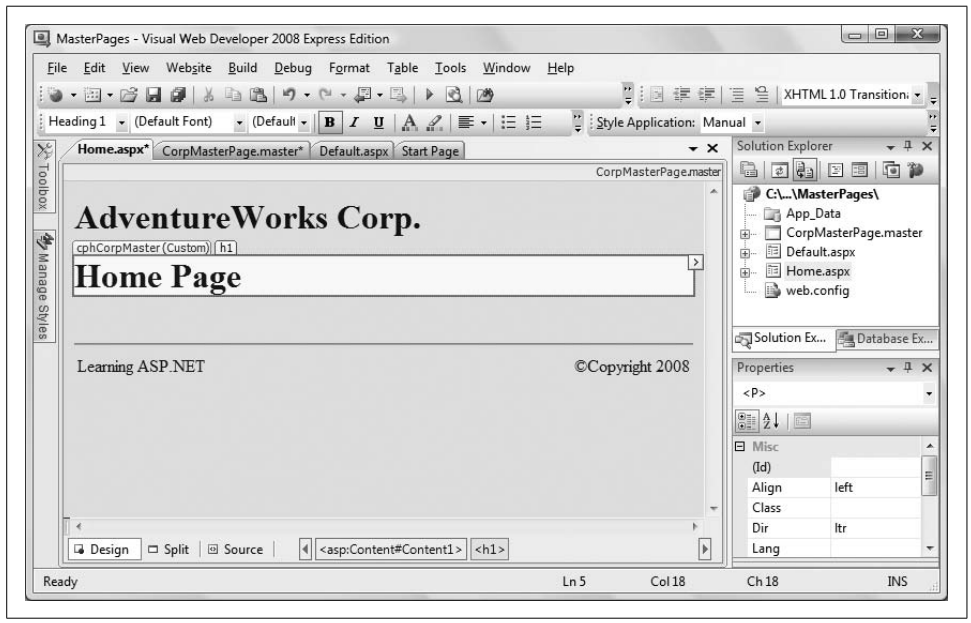


Figure 6-11. *When you create a content page, you add content within the placeholder you created before. The contents of the master page are there for you to view, but they're grayed out.*

The Design view allows you to see how your new page will look when it is combined with the master page at runtime.

The terminology can get a bit confusing, so let's clarify. A master page has an empty `ContentPlaceHolder` control.

You create a separate file called a content page. A content page is a normal *.aspx* file, with a `Page` directive but minus the `<html>`, `<form>`, `<head>`, and `<body>` tags, which are provided by the master page. Typically, you'll create many content pages for each master.

The contents of each content page are displayed as if they were inserted into the `ContentPlaceHolder` control. In effect, the Master Page is "wrapped around" the content page, allowing all the content pages to share the contents of the master page.

Create the next page, *SecondPage.aspx*, using the same master page. Using the master page ensures that the look and feel of the two pages will be identical.

Take a quick look at the markup generated for the second page:

```
<%@ Page Language="VB"
    MasterPageFile="~/CorpMasterPage.master"
    AutoEventWireup="false"
    CodeFile="SecondPage.aspx.vb"
    Inherits="SecondPage"
    title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="cphCorpMaster" Runat="Server">
</asp:Content>
```

The `Page` directive contains a reference to the master page for the second page, as well as some other information necessary to the page. An ASP.NET `Content` control was added for you automatically. Visual Studio knows which master page you are using automatically, and links the `Content` control to the `ContentPlaceholder` control on the master page.

You can put some simple text in the `Content` control and then run the two pages, as shown in Figure 6-12.

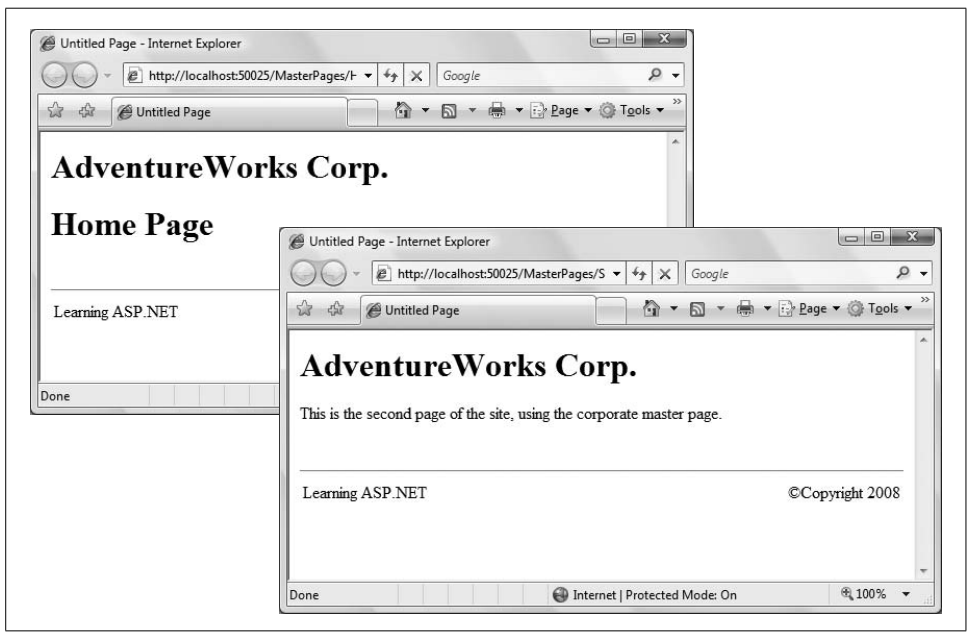


Figure 6-12. *This is what your content pages look like when you run the application. As you can see, the header and footer from the master page appear in both.*

This example does not provide any means of navigating from page to page. In order to see these pages, select the page in the Solution Explorer and run the web site. Later in this chapter, we will look at ways to navigate from page to page within a web site.

Using Nested Master Pages

You may want certain elements to appear throughout the entire web site, while other elements should be shared only within a specific part of your application. For example, you might have a company-wide header, but need division-specific elements as well. For situations like these, ASP.NET lets you create *nested master pages*. Any given web page can be combined with a nested master page or with the original master, whichever makes more sense for that individual page.

Copy the previous example, *MasterPages*, to a new application, *NestedMasterPages*.

Add a new master page to the web site, called *SalesMasterPage.master*. As shown back in Figure 6-8, be sure to check the "Place code in separate file" checkbox.

Look at the Source view; the IDE puts the following boilerplate markup code in *SalesMasterPage.master*:

```
<%@ Master Language="VB" CodeFile="SalesMasterPage.master.vb"
    Inherits="SalesMasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:contentplaceholder id="ContentPlaceHolder1" runat="server">
        </asp:contentplaceholder>
    </div>
    </form>
</body>
</html>
```

In order to make this a nested master page, delete *all* of this *except* for the `Master` directive. Then, add the highlighted code shown in Example 6-7. Again, notice how IntelliSense helps you.

Example 6-7. *Add some additional content to SalesMasterPage.master*

```
<%@ Master Language="VB" CodeFile="SalesMasterPage.master.vb"
   MasterPageFile="~/CorpMasterPage.master"
   Inherits="SalesMasterPage" %>

<asp:Content ID="SalesMasterContent" runat="server"
   ContentPlaceHolderID="cphCorpMaster">
   <table>
      <tr>
```

Example 6-7. *Add some additional content to SalesMasterPage.master (continued)*

```
        <td>
           <h3>Sales Department master page</h3>
           Put information here to display on all the Sales pages.
           <br />
           <br />
        </td>
     </tr>
     <tr>
        <td>
           <asp:ContentPlaceHolder ID="cphSalesContent" runat="server">
              Default content for Sales
           </asp:ContentPlaceHolder>
        </td>
     </tr>
  </table>
</asp:Content>
```

The `Master` directive has an additional attribute, `MasterPageFile`, which points to its own master page. This is how ASP.NET knows that this is a nested master page. In this way, master pages can be nested as deep as necessary.

This master page for the Sales department has an ASP.NET `Content` control, called `SalesMasterContent`, which contains the content to display on all the Sales pages. In this example, that content consists of an HTML table for layout, along with some additional markup.

Like all Content controls, it has a `ContentPlaceHolderID` attribute that specifies which `ContentPlaceHolder` control on *its* master page it will populate—in this case, `cphCorpMaster` on the *CorpMasterPage.master* master page.

The markup also includes a `ContentPlaceHolder` control called `cphSalesContent`. Content pages that use this nested master page will put their content inside this `ContentPlaceHolder`.

Switch to Design view for this nested master page, as shown in Figure 6-13. You can see the top-level master page, *CorpMasterPage.master*, except it is not editable, just as the master page of a normal content page is not editable in Design view.

To see the nested master page in action, add two new pages to the web site. Call them *Sales_Orders.aspx* and *Sales_Stores.aspx*. For each, check the "Select master page" checkbox shown previously in Figure 6-10. Now, when the Select a master page dialog comes up, you have two master pages to choose from. Select *SalesMasterPage.master* and click OK, as shown in Figure 6-14.

Add some content to each page to distinguish it. The markup for *Sales_Orders.aspx* is shown in Example 6-8.

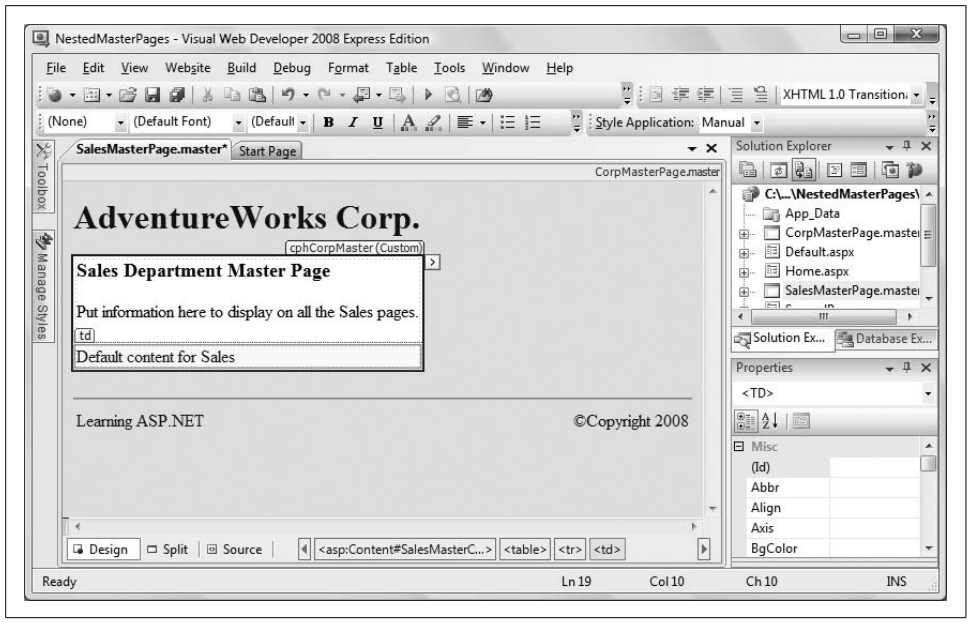


Figure 6-13. *When you view nested master pages in Design view, the child master page is editable, but the top-level master page is not.*

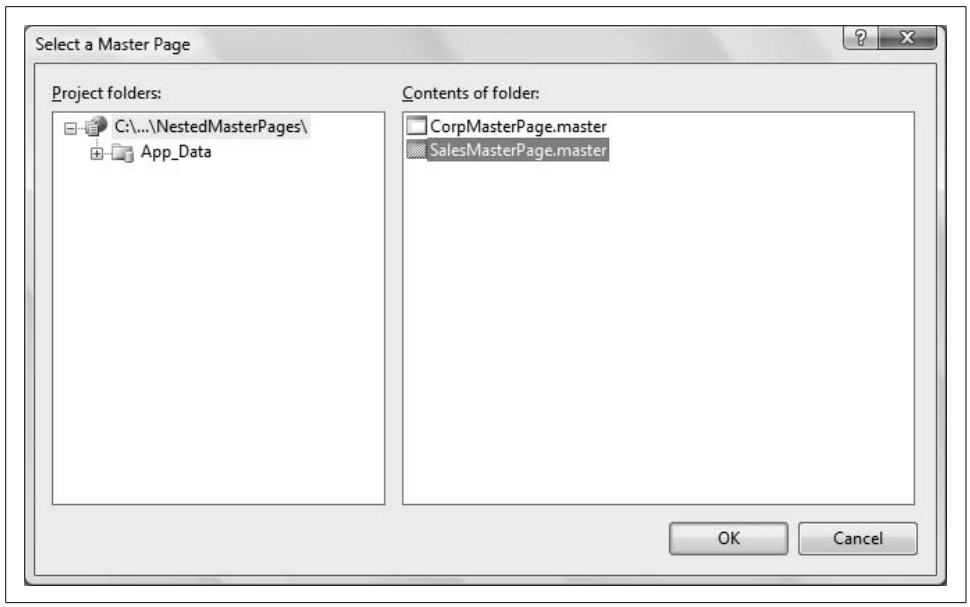


Figure 6-14. *When your site uses more than one master page, you have a choice whenever you add a new page.*

Example 6-8. *Sales_Orders.aspx is a simple content page to display with your master pages*

```
<%@ Page Language="VB" MasterPageFile="~/SalesMasterPage.master"
    AutoEventWireup="false" CodeFile="Sales_Orders.aspx.vb"
    Inherits="Sales_Orders" title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="cphSalesContent"
    Runat="Server">
    <h3>Orders</h3>
    Display Orders information here.
</asp:Content>
```

Place similar content in the *Sales_Stores.aspx* page, but change the text so that it reads "stores" instead of "orders."

The results for both pages are shown in Figure 6-15.

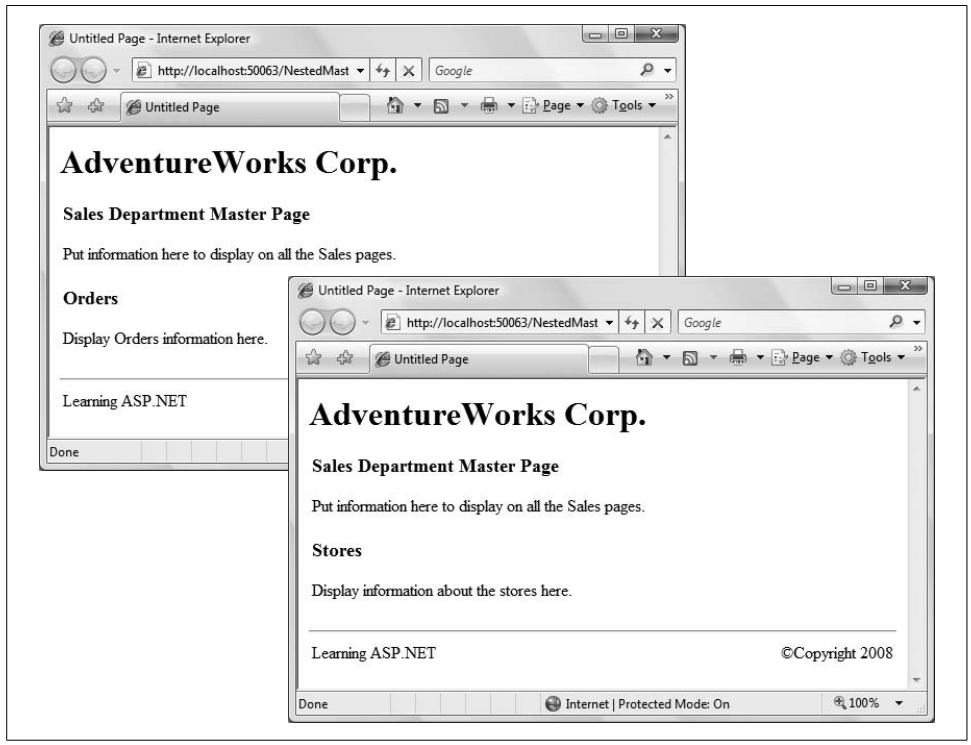


Figure 6-15. *This is what your nested master pages look like when you run the application. You can see the AdventureWorks master page—and the sales department master page—with the child content inside.*

Changing the Master Page at Runtime

You may decide that in response to certain events, you'd like to reach up into the master page (from a content page) and change its presentation. To do so, you need to add a *public* property in the master page that can be accessed by any of the content pages.

To see how this is done, copy the previous example MasterPages to a new web site called ChangingTheMasterPage.

In Source view, open the master page, *CorpMasterPage.master*. From the Standard section of the Toolbox, drag a `Label` control onto the page between the `<h1>` heading and the existing `ContentPlaceHolder` control. Change the `ID` of the Label to `lblMessage` and remove the `Text` property. Add an HTML line break and a horizontal rule (`<hr/>`) after the `Label` control while you are at it. The Source view of the master page markup will look like Example 6-9, with the new `Label` control and the additional HTML formatting highlighted.

Example 6-9. *CorpMasterPage.master with the Label control added*

```
<%@ Master Language="VB" CodeFile="CorpMasterPage.master.vb"
    Inherits="CorpMasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1>AdventureWorks Corp.</h1>
        <asp:Label ID="lblMessage" runat="server" ></asp:Label>
        <br />
        <hr />
        <asp:contentplaceholder id="cphCorpMaster" runat="server">
        </asp:contentplaceholder>
        <br />
        <hr />
        <table width="100%">
            <tr>
                <td width="50%" align="left">Learning ASP.NET</td>
                <td width="50%" align="right">&copy;Copyright 2008</td>
            </tr>
        </table>
    </div>
    </form>
</body>
</html>
```

If you run the web site now, this label will not be visible on any of the content pages because its Text property has no value.

Open the code-behind for the master page by right-clicking on *CorpMasterPage.master* in the Solution Explorer and selecting View Code. Type in the code highlighted in Example 6-10 to implement the public property.

Notice how IntelliSense helps you complete the code for the property.

Example 6-10. *CorpMasterPage.master.vb with the public property that will set the value of the label*

```
Partial Class CorpMasterPage
    Inherits System.Web.UI.MasterPage
Public Property MessageLabel( ) As Label
    Get
        Return lblMessage
    End Get
    Set(ByVal value As Label)
        lblMessage = value
    End Set
End Property

End Class
```

Before a content page can use a public property of the master page, it needs to be told the name of the class, or type, that contains the master page. This is done with another directive at the top of the markup of the page. You have already seen Page directives at the top of normal *.aspx* pages, as well as the Master directive at the top of the master pages. Now, add the following MasterType directive to the top of *SecondPage.aspx*, after the existing Page directive and before any page content:

```
<%@ MasterType TypeName="CorpMasterPage" %>
```

If you look at the code-behind for the master page *CorpMasterPage.master*, you will see that the name of the class is CorpMasterPage. This code, shown in the following snippet, was generated for you automatically by the IDE:

```
Partial Class CorpMasterPage
    Inherits System.Web.UI.MasterPage
```

Now that the content page has a reference to the class of the master page, it can access members of the master page in code. To see this, switch *SecondPage.aspx* to Design view. Drag a Button from the Standard section of the Toolbox into the Content section of the page. Change the ID of the Button to btnMessage and its Text property to Message Master. Figure 6-16 shows the page in Design view.

Double-click on the button to create an event handler for the Click event in the code-behind. Enter the highlighted code from Example 6-11 to this event handler to change the message label on the master page when the button is clicked.

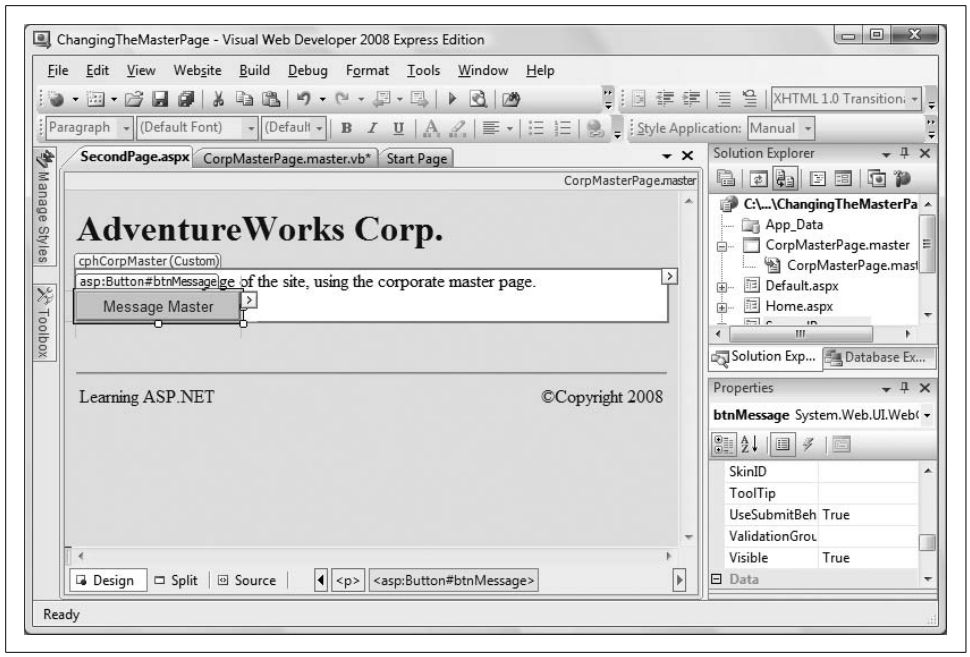


Figure 6-16. *You're adding a button to SecondPage.aspx that you're going to use to send a message to the master page.*

Example 6-11. *SecondPage.aspx.vb doesn't seem to do much, but it affects the label on the master page*

```
Partial Class SecondPage
    Inherits System.Web.UI.Page

Protected Sub btnMessage_Click(ByVal sender As Object, _
           ByVal e As System.EventArgs) _
           Handles btnMessage.Click
       Me.Master.MessageLabel.Text = "Button on SecondPage pushed."
End Sub
End Class
```

Run *SecondPage.aspx.vb* and click the button. The result will look similar to Figure 6-17.

Navigation

Modern commercial web sites can be surprisingly complex, often consisting of dozens, even hundreds of pages. Users will have a more satisfying experience if you provide navigational hints and menus to help them avoid getting lost, and to enable them to conveniently find all the features of the site.

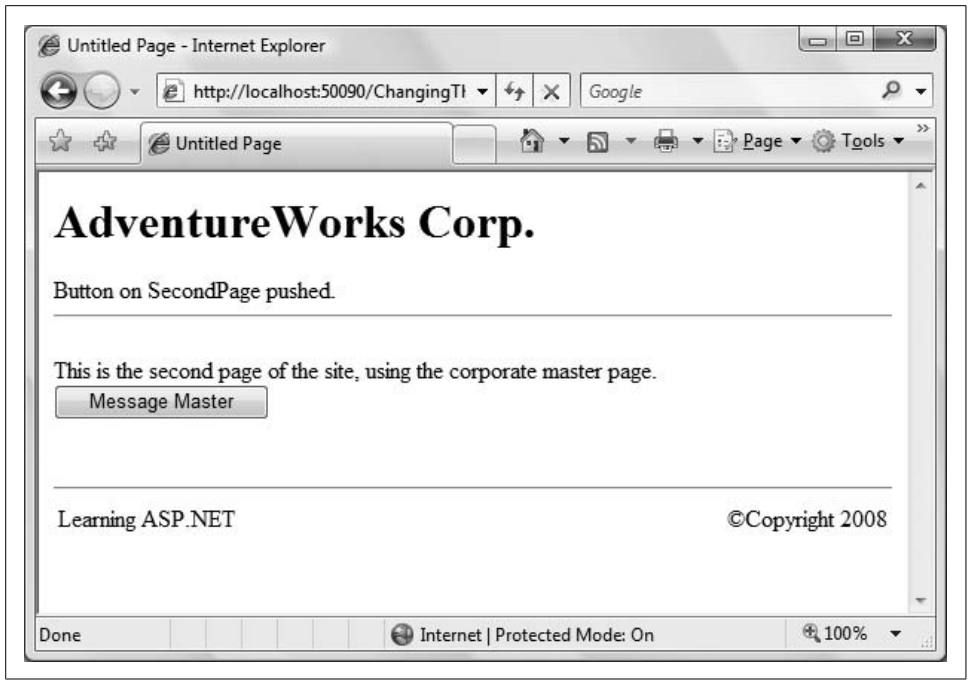


Figure 6-17. *When you click the button on SecondPage, the message is sent to the master page and displayed.*

The ASP.NET toolset includes a number of controls that facilitate this assistance. There are controls for creating both "bread crumbs" (how did I get to this page?) and site maps (how do I find that other page?).

Often you will want these features to be present on every page, and thus master pages are a great asset. If you change the site map or the control, you only need to update the master page, and all the other pages are automatically current.

Buttons and HyperLinks

The simplest form of navigation is through the use of `Buttons`, `LinkButtons`, and `Hyperlinks`. All three will take the user to a different page. Superficially, `LinkButtons` and `Hyperlinks` look the same, while buttons look different (see Figure 6-18). Under the covers, however, `LinkButtons` and `Buttons` have much more in common, while `Hyperlinks` are very different.

Let's clarify. When you click a `Hyperlink`, you are taken directly to a new page. The first page does not post back to the server. You are immediately transferred to the new page—do not pass Go, do not collect $200.

VB CHEAT SHEET

Access Modifiers

You've seen lots of properties throughout this book. For example, most controls have a `Text` property. You have also seen the use of local variables in the classes defined in the code-behind files. The declaration for properties and variables include a keyword known as the *access modifier*, which specifies what parts of the program can see that property or variable. The most common access modifiers are `public`, `private`, and `protected`.

If a class member is declared public, that member is available anywhere in the application. All you have to do to access the public member is declare an instance of that class. On the other hand, if a class member is declared private, then only the code within the class itself can access that member.

To put this into relevant terms, in Example 6-10, the property `MessageLabel` has declared `public`. Code outside the class (outside the master page) can refer to that property by referring to an instance of the class and the property using dot notation, as will be demonstrated below. If the property were `private`, only code in the `CorpMasterPage` class would be able to access it, which would defeat the purpose here because we want other pages to be able to access this property. If it were `protected`, only `CorpMasterPage` and classes that inherit from `CorpMasterPage` could use the property.

If a member is public, then it will be displayed by IntelliSense where appropriate. If it is private or protected, IntelliSense will never display it if you're writing code outside the class.

If you do not declare an access modifier, the default is public, but it is always good practice to explicitly declare it, even if public is what you intend.

With a `LinkButton` (which looks like a `Hyperlink`) or a `Button` (which looks like a button), however, the page *is* posted back, and there is an opportunity for you, the developer, to run an event handler *before* control is handed over to the new page.

`Hyperlinks` are faster for the user (and simpler for the developer), but they do not give you the opportunity to run code before the user leaves the page, which is the trade off you'll have to make each time you decide between a `Hyperlink` and one of the alternatives.

To see how buttons and links can be used for navigation, copy the example from earlier in this chapter, MasterPages, to a new web site called ButtonNavigation.

Open *Home.aspx* in Design view. Drag a Button control onto the page below the header. Change the `Text` property of the Button to "Page 2". It will look something like Figure 6-19.

VB CHEAT SHEET

The Me Object

So far, your event handlers have always affected the properties of controls on the same page. In this case, however, you want to change the property of a control on the master page. *SecondPage.aspx.vb* doesn't know where that control is. Fortunately, every content page does know its own master. When you want a class to call a method on itself, or access one of its own properties, you use the `Me` keyword. `Me`, as you might expect, refers to the class that's calling the method or accessing the property. In this case, you want to access the master page of *SecondPage.aspx.vb*, so you do that with a reference to `Me.Master`. From there, it's easy to access the `MessageLabel.Text` property on the master page. You don't strictly need to use the `Me` keyword in this case; if you leave it out, the code works just fine. However, it is good practice to use `Me` so that people reading your code will know what you're referring to.

The equivalent object in C# is called `this`.

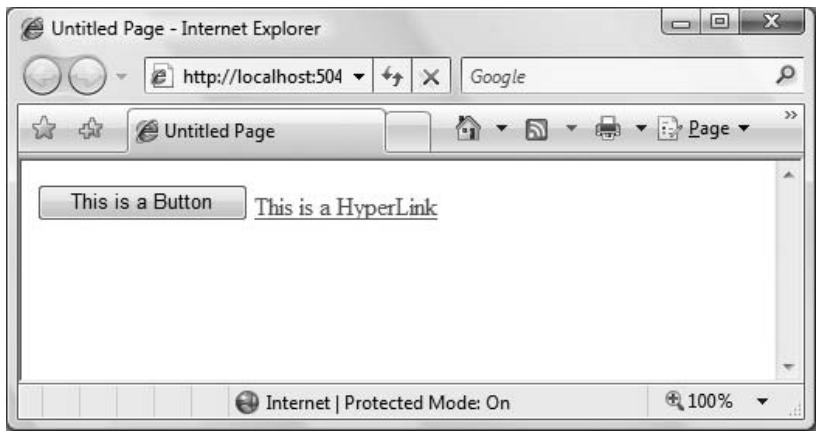


Figure 6-18. *A Button control looks like you would expect a button to look, and ditto for the HyperLink control.*

As previously mentioned, you need to add custom code to make the navigation happen. You'll need to provide some code to handle the `Click` event, so in Design view, double-click the button and then enter the highlighted lines of code shown in Example 6-12.

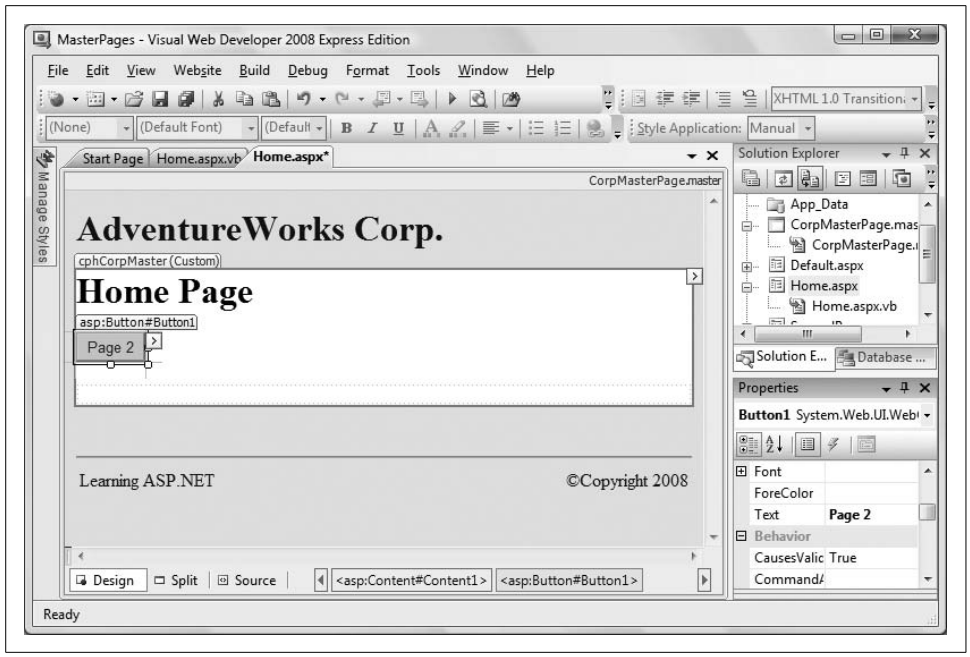


Figure 6-19. *You've placed a navigation button on the home page, which will help users find where they're going.*

Example 6-12. *Home.aspx.vb showing theButton Click event handler*

```
Partial Class Home
    Inherits System.Web.UI.Page

Protected Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
    ' do something here
     Response.Redirect("SecondPage.aspx")
End Sub
End Class
```

Before the button event handler redirects to the new page, you can run some other code. You might do so, for example, to retrieve the status of other controls on the page, interact with a database, perform computations or, very commonly, to stash a value in Session State that will be retrieved by the new page. (We'll discuss Session state in Chapter 7; you don't need to worry about it for the moment.)

The Button's `Click` event handler is not the only place where your code is executed during postbacks. You can place code in event handlers for any number of events. By far, the most common is Page Load, where it is routine to place code to process the page. Page Load and other life cycle issues are covered in Chapter 8.

The actual navigation is accomplished with the Redirect method of the HttpResponse class. It is the programmatic equivalent of a hyperlink, immediately transferring to the new page without first posting back to the server. The argument to the method is a string representing the URL of the target page.

It is somewhat confusing that the syntax for calling the Redirect method of the HttpResponse class is:

```
Response.Redirect( )
```

and not:

```
HttpResponse.Redirect( )
```

but there you have it. This oddity is for historical reasons.

The URL can either be relative (as in this example) or absolute. In this example, it refers to a web page in the same directory as the current page. An absolute URL would be completely qualified, regardless of the current location, such as *http://LibertyAssociates.com/Samples/SecondPage.aspx*.

Run the page now to see how it works. Clicking on the button posts the page back to the server. If there were a method called Page_Load to handle the Page Load event, it would be executed. Then, the code in the Button Click event handler from Example 6-12 would run. The last line in that method would be the Response.Redirect to perform the navigation.

The main attraction to using buttons and links for navigation is that they are very simple and direct. The big problem is that it can be tedious to implement, as you must place every button or link on every page, specify the URL, and handle the Click event for each (for Buttons and LinkButtons). As a case in point, this example so far allows you to navigate from the home page to the Second page, but not back (without using the browser's Back button). For a web site with many pages and routes, this approach quickly breaks down.

At any rate, add a HyperLink control to *SecondPage.aspx* to allow easy navigation back to the home page. Go to *SecondPage.aspx* in Design view, hit the Enter key a few times at the end of the line of text already there then drag a HyperLink control on to the page. In the Properties window, set the Text property to Home Page, and the NavigateUrl property to Home.aspx. Figure 6-20 shows the home page link in Design view.

Now, if you run the page, there will be hyperlink on the second page to take you back to the home page. In this example, they behave identically, but if you needed custom code to execute on *SecondPage.aspx*, that would not happen with the HyperLink.

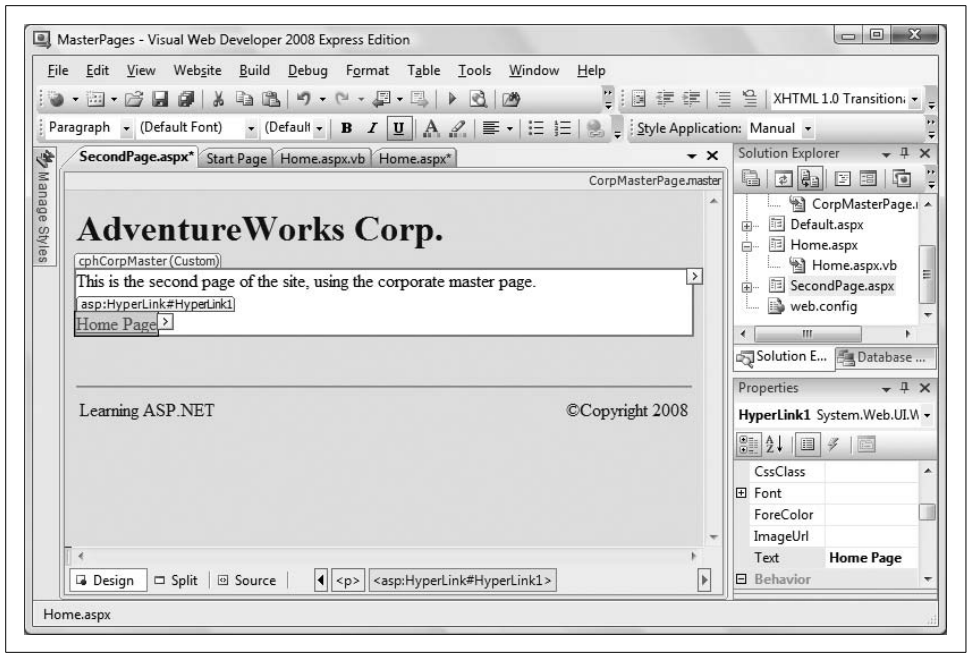


Figure 6-20. *A HyperLink in Design view showing the properties set to give it a target page to navigate to.*

Menus and Bread Crumbs

You've probably seen menus and bread crumbs for navigation on many commercial sites. Menus are familiar from the earliest windowing environments; they offer a set of choices for navigation (they can be pull-down or pop-up selections). *Bread crumbs* take their name from the story of Hansel and Gretel, who left a trail behind them so they could find their way home. In ASP.NET, bread crumbs typically consist of a set of links back through the web site, making it easy for users to reverse course and take different forks in what otherwise would be a confusing tree of alternative pages.

To see menus and bread crumbs at work, you'll need a web site with a few pages to simulate a complex web site of hundreds of web pages (feel free to create hundreds of web pages if you like; we'll wait). Figure 6-21 shows how the finished web site will appear.

To build this web site, you will use a single master page, several normal web pages, and a site map to provide information for the menu and the bread crumbs. Later in the chapter, you will see how to spiff up the appearance of the menu and the bread crumbs.

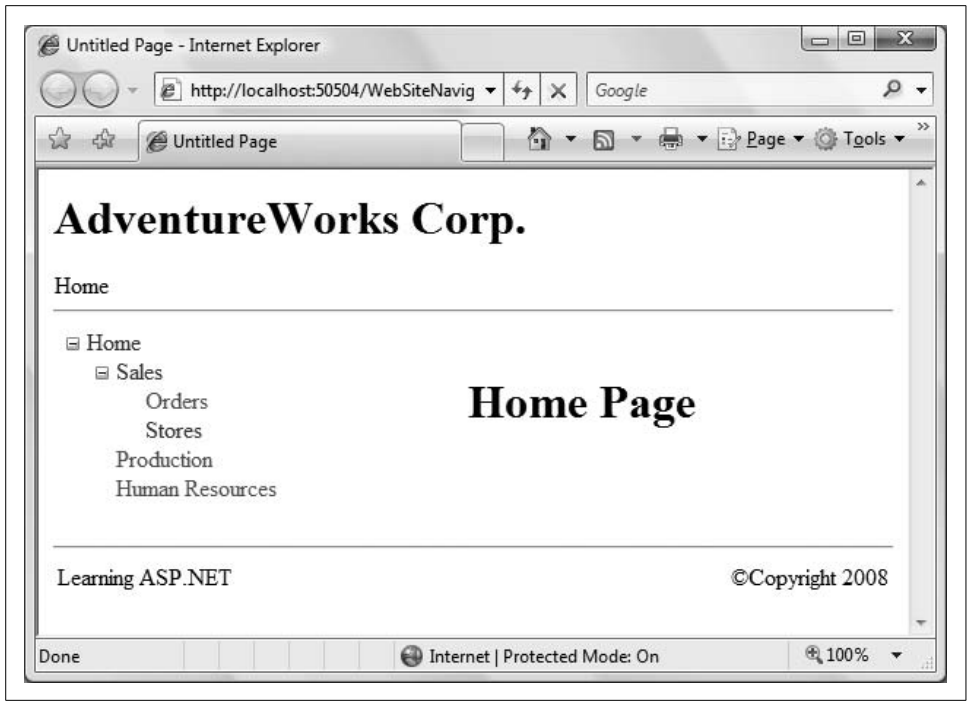


Figure 6-21. *When you've completed the web site with menus and bread crumbs, it will look like this.*

First, you will create the web site with the master page, minus the navigation controls, and all the content pages. Next, you will add the site map and the navigation controls.

Create a new web site called WebSiteNavigation. Close the *Default.aspx* page if it was opened when you created the site. Then, delete *Default.aspx* by clicking on it in the Solution Explorer and pressing the Delete key. Confirm the deletion by clicking OK.

Add a master page, exactly as you did previously in this chapter. Make certain that you have the root of the website selected in the Solution Explorer, and click on Website → Add New Item…. In the Add New Item dialog box, select master page. You can retain the default file name of *MasterPage.master*. Be sure to select the "Place code in separate file" checkbox (see Figure 6-8, earlier in this chapter).

When the master page opens in Source view, the only markup inside the `<div>` elements is the default `ContentPlaceHolder` control. Replace that with the highlighted code shown in Example 6-13, and also delete the `ContentPlaceHolder` control inside the `<head>` elements. This is nearly identical to the *CorpMasterPage.master* from the example shown earlier in the chapter, except the `ContentPlaceHolder` control is

placed inside an HTML table for layout control. Notice that the first cell in the only row in that table is empty at the moment. You will put the menu in that cell shortly.

Example 6-13. *This is what MasterPage.master looks like before you add the navigation controls*

```
<%@ Master Language="VB" CodeFile="MasterPage.master.vb"
      Inherits="MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Untitled Page</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1>AdventureWorks Corp.</h1>
      <br />
      <hr />
      <table width="100%">
         <tr>
            <td></td>
            <td>
               <asp:ContentPlaceHolder ID="cphCorpMaster" runat="server">
               </asp:ContentPlaceHolder>
            </td>
         </tr>
      </table>
      <br />
      <hr />
      <table width="100%">
         <tr>
            <td width="50%" align="left">Learning ASP.NET</td>
            <td width="50%" align="right">&copy;Copyright 2008</td>
         </tr>
      </table>
   </div>
   </form>
</body>
</html>
```

As you can see from the menu in Figure 6-21 shown previously, there are six pages in this web site. Add all six of those pages to the web site now. In all cases, be sure to check the "Select master page" checkbox (see Figure 6-10, earlier in this chapter), and to select *MasterPage.master* as the master page (the only choice). The names of the pages to create are:

- *Home.aspx*
- *HR.aspx*
- *Production.aspx*

- *Sales.aspx*
- *Sales_Orders.aspx*
- *Sales_Stores.aspx*

To keep things simple, add only an `<h1>` heading to the Content area of each page identifying the name of the page.

You are now ready to prepare the site map and add the navigation controls.

Site Maps

Site maps are used as a data source for navigation controls such `TreeViews`, `Menus`, and `SiteMapPaths` (which provide bread crumbs). Used in conjunction with master pages, these allow for easy, central navigation without having to place navigation controls on every page of the web site.

Add a site map to the current web site by clicking on Website → Add New Item…. When the Add New Item dialog appears, select Site Map and accept the default name, *Web.sitemap*, as shown in Figure 6-22.

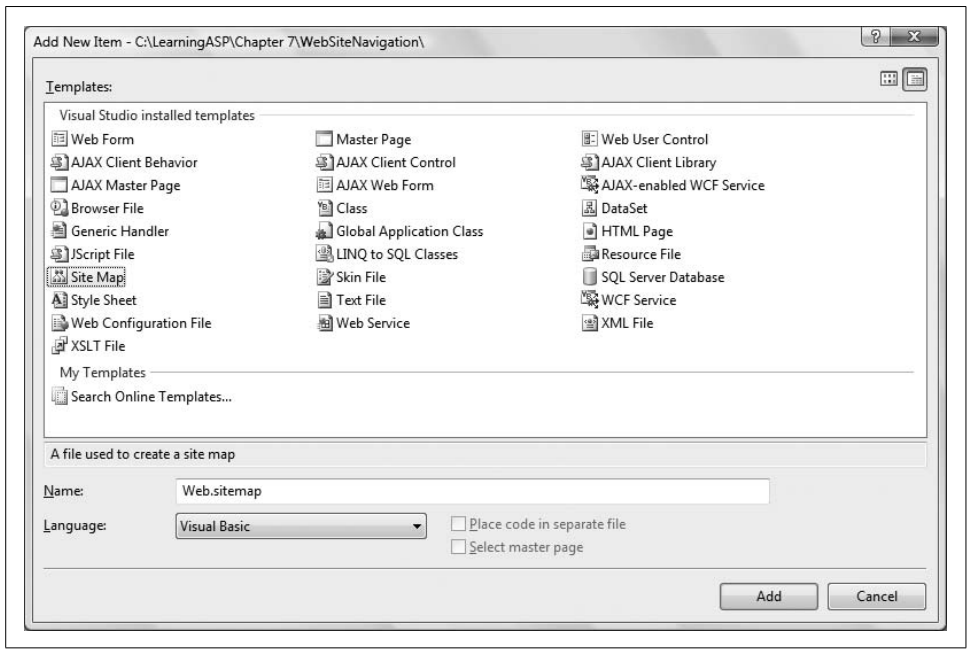


Figure 6-22. *To create a new Site Map, select Website → Add New Item, and then choose Site Map.*

When you click Add, the file *Web.sitemap* is added to your web site, and the skeleton of a site map is provided for you, as shown in Example 6-14.

Example 6-14. *When you choose to add a new site map, Visual Studio creates this Web.sitemap skeleton for you*

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="" title="" description="">
        <siteMapNode url="" title="" description="" />
        <siteMapNode url="" title="" description="" />
    </siteMapNode>
</siteMap>
```

The `url` attribute specifies the page this navigation link will point to. The `title` attribute defines the text that is displayed as the link, and the `description` attribute is used in the tool tip.

Neither VWD nor VS2008 provide drag-and-drop support for creating your site map file. You can implement your own SiteMap provider to automate this process, or get the site map from another source (such as a database) but that is a very advanced topic beyond the scope of this book.

Replace the contents of *Web.sitemap* with the site map XML shown in Example 6-15.

Example 6-15. *Visual Studio created the skeleton of the Web.sitemap, but you need to provide the actual content*

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="~/Home.aspx" title="Home" description="Home page">
        <siteMapNode url="~/Sales.aspx" title="Sales" description="Sales">
            <siteMapNode url="~/Sales_Orders.aspx" title="Orders"
                description="Orders" />
            <siteMapNode url="~/Sales_Stores.aspx" title="Stores"
                description="Stores" />
        </siteMapNode>
        <siteMapNode url="~/Production.aspx" title="Production"
                description="Production" />
        <siteMapNode url="~/HR.aspx" title="Human Resources" description="HR" />
    </siteMapNode>
</siteMap>
```

The site map is an XML file, as indicated by the first line in the file. The hierarchy of the web site is represented by a set of nested `<siteMapNode>` elements. The root element is a single `<sitemap>` element that defines the namespace:

```
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
```

Within the `siteMap` element is nested exactly one `<SiteMapNode>` (in this case, Home). Nested within that first `<SiteMapNode>`, however, is any number of children `<SiteMapNode>` elements. Each `<SiteMapNode>` element can in turn have any number of children `<SiteMapNode>` elements.

In Example 6-15, there are three such children: Sales, Production, and Human Resources. Nested within each of these `<SiteMapNode>` elements can be more nodes. For example, Sales contains Orders and Stores. You may nest the nodes as deep as you wish.

ASP.NET is configured to protect files with the extension *.sitemap* so they cannot be seen by a browser.

Once the site map file is in place, you need to make it available to the master page. You do that by dragging a `SiteMapDataSource` control from the Data section of the Toolbox onto the master page. By default, the `SiteMapDataSource` control will look for and use the file named *Web.sitemap*.

It doesn't matter where you place this `SiteMapDataSource` control, as long as it is somewhere between the `<form>` and `</form>` tags in Source view. The `SiteMapDataSource` will be visible in Design view but will not appear when the web site is run.

The Design view should look something like Figure 6-23.

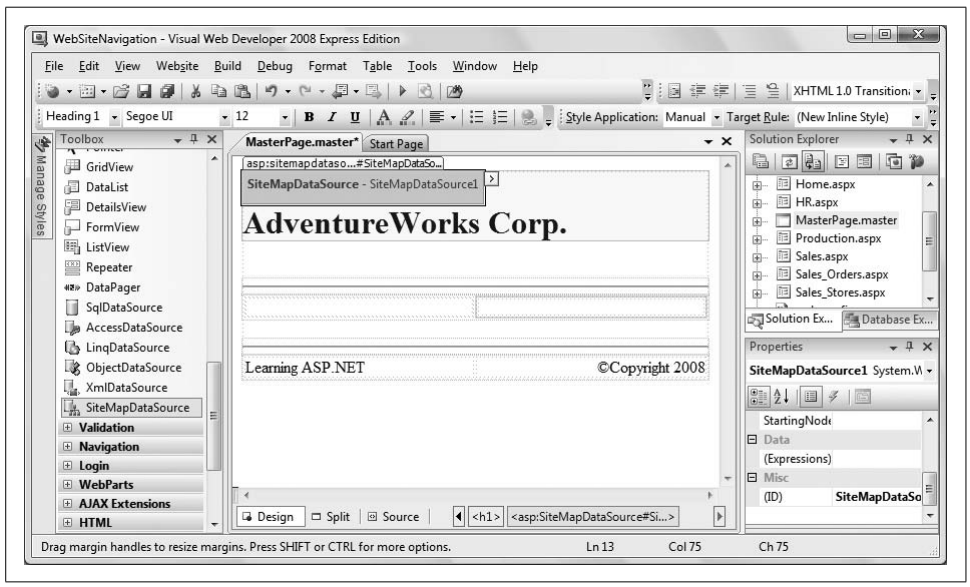


Figure 6-23. *Place your SiteMapDataSource control on the master page. It'll show up in Design view, but not in any of the pages that use the master page.*

Using Sitemaps

There are two types of controls that can read a site map and provide navigation capability: a `TreeView` control and a `Menu` control. To see both at work, you'll first create a `TreeView` control, then you will disable that and add a Menu control.

TreeView

The `TreeView` provides the familiar hierarchical view of items. One common use of a `TreeView` is the Windows Explorer view of folders and subfolders.

Recall when you created the master page for this example there was an empty table cell in the layout. Switch to Source view, and then, from the Navigation section of the Toolbox, drag a `TreeView` control into that empty cell. (You can do this in Design view, of course, but we find it easier to use Source view when dragging elements into a cell.)

Switch back to Design view and click on the Smart Tag of the `TreeView`. Click on the drop-down next to Choose Data Source, and select `SiteMapDataSource1`, the `ID` of the `SiteMapDataSource` you just placed on the master page, as shown in Figure 6-24.

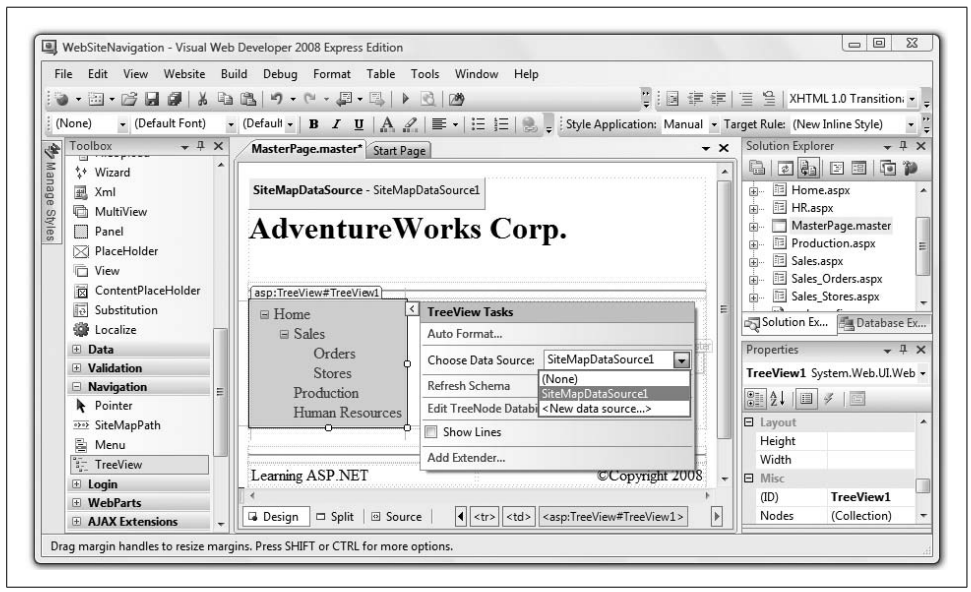


Figure 6-24. *Select the data source for the TreeView control from the Smart Tag. In this case, you just have the one SiteMapDataSource control on the master page.*

As soon as you select the data source for the `TreeView`, it will display the nodes from the site map file in Design view.

Set *Home.aspx* as the startup page by right-clicking on it in the Solution Explorer and selecting "Set As Start Page" from the menu.

Run the web site now and see the TreeView menu in action. Click on any of the menu items along the left of the web site and observe how it navigates from page to page. If you click on the Stores menu item, it will look almost exactly like Figure 6-21 from earlier in this chapter. The only difference is the bread crumbs are missing because you have not yet placed that control on the page.

Menu items that contain subitems of their own, such as Home and Sales, display a small icon next to them. Clicking on this icon toggles between expanded and collapsed views of these subitems. This structure directly flows from the nesting of SiteMapNodes in the site map file.

Customizing the look and feel of the TreeView

The TreeView control has many properties, methods, and events that allow you to customize the look and feel of the TreeView.

Adjusting the Root Node

If you don't like the way the root node sticks out, you can adjust it. Set the ShowStartingNode property of the SiteMapDataSource control to False, then edit the *Web.sitemap* file so that the original root node is now one level in, and an empty root node takes its place, as in the following:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/
AspNet/SiteMap-File-1.0" >
    <siteMapNode title="Root" >
        <siteMapNode url="~/Home.aspx" title="Home"
                     description="Home page" />
        <siteMapNode url="~/Sales.aspx" title="Sales"
                     description="Sales" >
            <siteMapNode url="~/Sales_Orders.aspx"
                     title="Orders"
                     description="Orders" />
            <siteMapNode url="~/Sales_Stores.aspx"
                title="Stores"
                description="Stores" />
        </siteMapNode>
        <siteMapNode url="~/Production.aspx"
                title="Production"
                description="Production" />
        <siteMapNode url="~/HR.aspx"
                title="Human Resources"
                description="HR" />
    </siteMapNode>
</siteMap>
```

The only attribute actually required of the root node is the Title.

The easiest way to change the appearance of the `TreeView` is to view the page in Design view, click the `TreeView`'s Smart Tag, and then click Auto Format…, as shown in Figure 6-25.

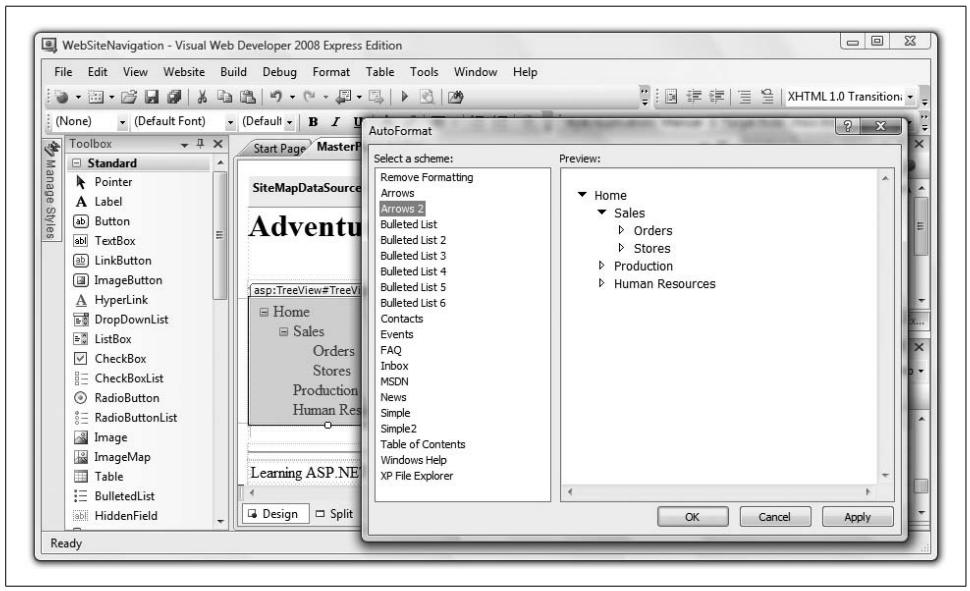


Figure 6-25. *The Auto Format options, found in the Smart Tag of the TreeView, provide a number of prebuilt formatting options.*

Most of the `TreeView`'s properties have to do with the styles used for the various nodes. There are properties for general nodes, parent nodes, child (leaf) nodes, root nodes, selected nodes, and nodes when the mouse is hovering over them. For each of these node types you can set font attributes, CSS class, fore- and back-colors, spacing and padding, borders, and so on.

Replacing the TreeView with a menu control

Open *MasterPage.master* in Source view and locate the `TreeView` control. Comment it out with HTML comment tags and replace it with a `Menu` control. Remember to set the Data Source, either in the Smart Tag or directly in Source view:

```
<!--<asp:TreeView ID="TreeView1" runat="server"
          DataSourceID="SiteMapDataSource1"
     </asp:TreeView> -->
 <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1" />
```

Run the application. Presto! A menu control for navigation. Hover over Home (opening the next level), and then hover over Sales (opening the third level). Finally, click Stores. The results should look like Figure 6-26.

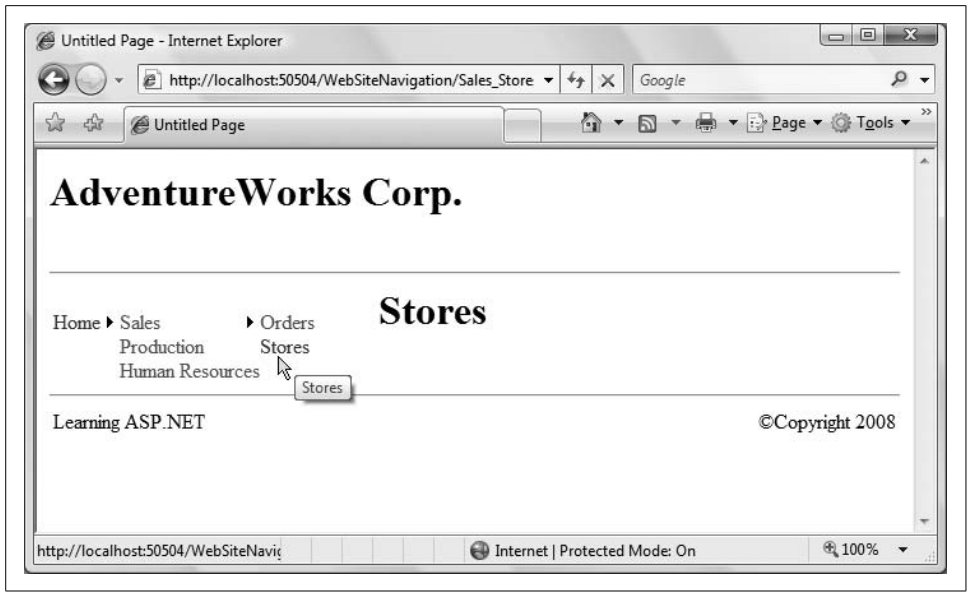


Figure 6-26. *You've replaced the TreeView with a menu control, which does all the work for you. When you hover over Stores in the Menu Control, you can navigate to that page.*

If the menus start to eat into your content space, you can set their `Orientation` property to `Horizontal` (the default is `Vertical`) and rearrange your layout table to make room for them.

Accessing site map nodes programmatically

There are times when you may want access to the current node and its subnodes so you can manipulate them programmatically. For example, you may want to log the user's current menu choice to a log file. You can add code to a page to get that information. In the next example, you will display the names of the current node and its subnodes in the *Sales.aspx* page. First, switch to the master page and remove the menu control you added in the previous example, and uncomment the `TreeView`. Now, switch to *Sales.aspx*, and add the highlighted code in Example 6-16 inside the `Content` tags, including the `<hr/>` to provide a bit of a visual break.

Example 6-16. *Sales.aspx with added Label controls for display of the current and child nodes*

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
   CodeFile="Sales.aspx.vb" Inherits="Sales" Title="Untitled Page" %>

<asp:Content ID="Content1" ContentPlaceHolderID="cphCorpMaster"
      runat="Server">
   <h1>
      Sales</h1>
   <hr />
   <table>
```

Example 6-16. *Sales.aspx with added Label controls for display of the current and child nodes*

```
      <tr>
         <td>
            <b>Current Node:</b>
         </td>
         <td>
            <asp:Label ID="lblCurrentNode" runat="server" />
         </td>
      </tr>
      <tr>
         <td valign="top">
            <b>Child Nodes:</b>
         </td>
         <td>
            <asp:Label ID="lblChildNodes" runat="server" />
         </td>
      </tr>
   </table>
</asp:Content>
```

You have added two labels, `lblCurrentNode` and `lblChildNodes`, but they have nothing to display yet. For that, you'll need an event handler.

Open the code-behind for this page (click the plus next to *Sales.aspx* in Solution Explorer, then double-click *Sales.aspx.vb* that appears below it). Add the highlighted code in Example 6-17 to create an event handler for the Page Load event. You can have the IDE create the skeleton of the event handler for you by selecting (Page Events) from the drop-down at the top left of the editing window and selecting Load from the drop-down menu at the top right of the editing window.

Example 6-17. *Sales.aspx.vb showing thePage_Load event handler*

```
Partial Class Sales
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, _
                ByVal e As System.EventArgs) Handles Me.Load
        Try
             Me.lblCurrentNode.Text = SiteMap.CurrentNode.Title
             If SiteMap.CurrentNode.HasChildNodes Then
                For Each node As SiteMapNode In SiteMap.CurrentNode.ChildNodes
                    Me.lblChildNodes.Text += node.Title + "<br />"
                Next
            End If
         Catch exNull As System.NullReferenceException
            Me.lblCurrentNode.Text = "The XML file is not in the site map!"
        Catch ex As System.Exception
            Me.lblCurrentNode.Text = "Exception! " + ex.Message
        End Try
    End Sub

End Class
```

In this code, you are setting the Text property of lblCurrentNode to reflect the Title property of the SiteMap's CurrentNode. The SiteMap is an in-memory representation of a site's navigational structure. The SiteMap object itself is created by the site map provider (in this case, by the SiteMapDataSource).

VB CHEAT SHEET

For Each

As you've seen throughout this book, lots of objects contain collections of other objects. Frequently, you'll want to take some action on every object in a collection, but you won't know how many there are. That's where the For Each loop comes in.

In this case, the SiteMap has a collection called ChildNodes, which is a collection of SiteMapNode objects. You want to grab each node in order, extract the Title property, and add it to a label. Here's how the For Each loop breaks down:

```
For Each node As SiteMapNode In
    SiteMap.CurrentNode.ChildNodes
```

You start with the For Each statement, and then you define a variable, node, which is of type SiteMapNode. Like all variables, node is a placeholder; in this case, you're using node to indicate "the node I'm looking at right now." You use As to indicate that node is of the type SiteMapNode because that's the kind of objects that ChildNodes contains.

Then, you use In to indicate where the loop should find the SiteMapNode objects to use, which in this case is inside SiteMap.CurrentNode.ChildNodes. When the loop starts, the first node from the ChildNodes collection gets loaded into node. You take an action on node, in this case, extracting the title and adding it to the label, followed by a line break:

```
Me.lblChildNodes.Text += node.Title + "<br />"
```

You could take more than one action, of course, assuming you want that action to be repeated multiple times.

The loop ends with Next. When the loop reaches that point, it dumps the current content of node, and repeats the loop on the next SiteMapNode object from ChildNodes. When the loop has gone through each item in the collection, it stops, and execution of the code continues from after the loop.

The CurrentNode property returns an object of type SiteMapNode, and the Title property of that SiteMapNode returns the title of that SiteMapNode.

The SiteMapNode's property HasChildNodes returns a Boolean, which is True if there are subnodes to the SiteMapNode. If this is the case, you can iterate through the SiteMapNodeCollection returned by the ChildNodes property. If there are no child nodes, this code does nothing.

VB CHEAT SHEET

Catching Errors

In Example 6-17, you can see the code that sets the text of your labels is contained within a block named `Try`. This is how VB accounts for unexpected errors. If an error occurs inside the `Try` block (known as *throwing* an exception), you can take action specific to the error by putting error-handling code in a `Catch` block. In this case, there are two `Catch` blocks: the first one occurs if an error is thrown of type `System.NullReferenceException`, meaning that the page doesn't have a valid reference to the XML file that stores the site map. If that happens, `lblCurrentNode` outputs an error message specific to the problem. The second `Catch` block is a general case, invoked for any error of type `System.Exception`. In this case, `lblCurrentNode` displays a general error message, and adds the `Message` property of the captured error, which is automatically generated by ASP.NET. Neither message fixes the problem, but at least the user will know what happened if an exception is thrown. In a real-world web site, you might have some code that supplies a substitute object, logs the error, or takes some other action to allow the user to continue.

Notice that the `Try` block ends with an `End Try` statement, which comes after all the `Catch` blocks.

When you view this page, the labels display the name of the current node and all its child nodes, as shown in Figure 6-27.

Bread Crumbs

The final thing to add to your example is bread crumbs. As we mentioned earlier, bread crumbs are an indicator of where you are in the page hierarchy and how you got there. This is done using the ASP.NET `SiteMapPath` control.

To see this, go back to *MasterPage.master* in Design view. From the Navigation section of the Toolbox, drag a `SiteMapPath` control onto the page between the AdventureWorks heading and the horizontal rule, as shown in Figure 6-28.

That's all there is to it!

Run the site and you'll see how the breadcrumbs tell you where you are at all times.

It is uncommon in production applications to provide both a `TreeView` and bread crumbs on the same page.

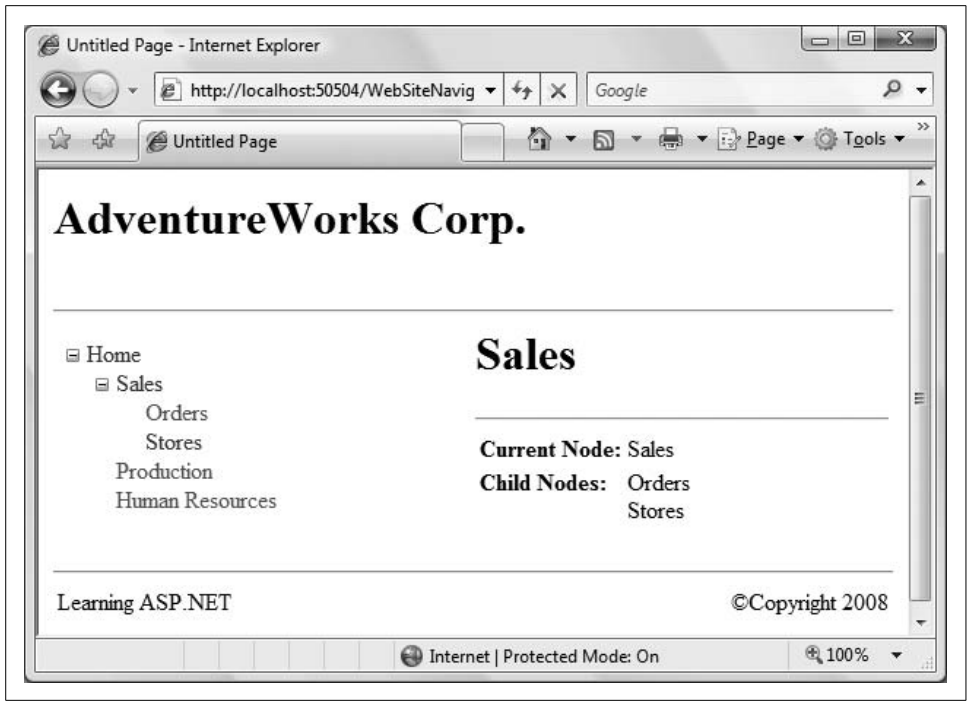


Figure 6-27. *You've added the code that allows you to access the current node and the child nodes, and that lists them for the user to see.*

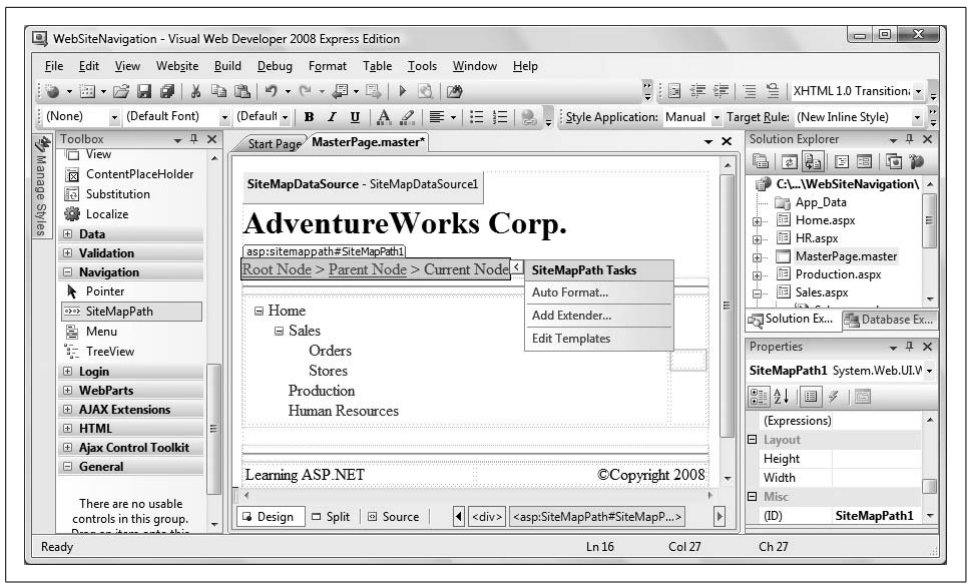


Figure 6-28. *Adding navigation with bread crumbs to your site is as simple as placing a SiteMapPath control in Design view.*

Similar to the `TreeView`, the `SiteMapPath` provides many ways to customize the look and feel. Click on the Auto Format item in the Smart Tag shown in Figure 6-29 to see a number of predefined formats. Alternatively, the Properties window provides a similar, if smaller, set of properties, just as it did for the `TreeView`.

In the previous example, the bread crumbs separated the various pages with the greater-than symbol (>). This is easy to change with the `PathSeparator` property. For example, to use an arrow as the separator symbol, edit the `SiteMapPath` control in Source view to look like the following:

```
<asp:SiteMapPath ID="SiteMapPath1" runat="server" PathSeparator="->" />
```

The result is shown in Figure 6-29. Compare this with the original bread crumbs shown earlier in Figure 6-21.

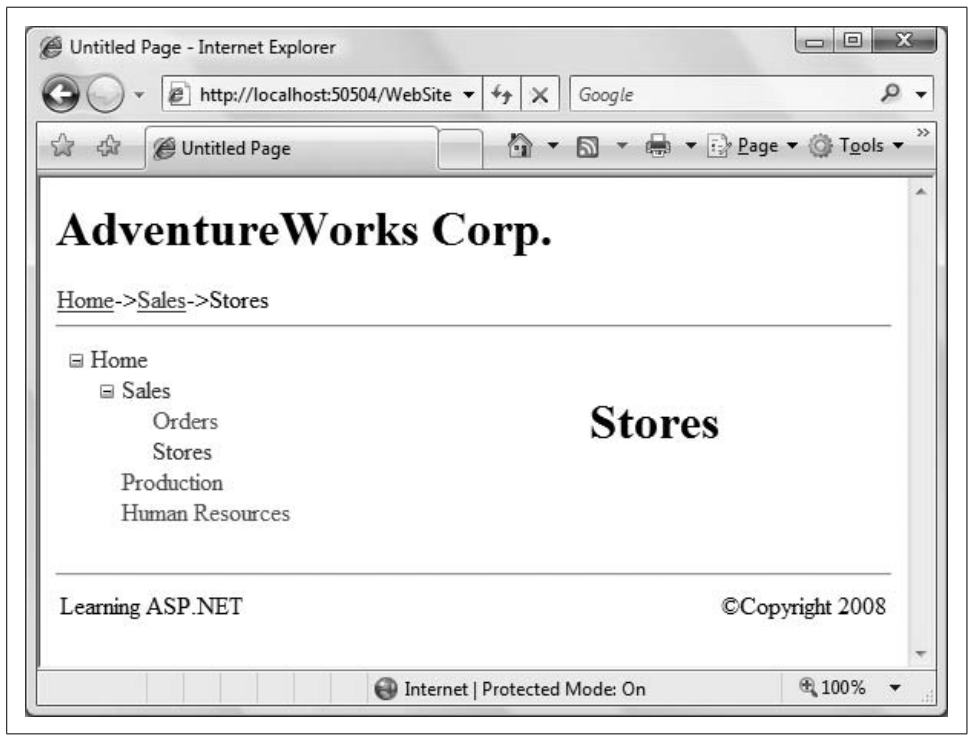


Figure 6-29. *You can change the PathSeparator property of your SiteMapPath control to alter the look of your bread crumbs, in this case, with an arrow symbol.*

Summary

- Style sheets hold presentation information for an entire site in a separate file to ensure a consistent look throughout the site.
- A style specifies how a specific element is rendered visually in the browser.
- Style rules are applied in a hierarchical manner, such that more specific styles take precedence over general styles.
- A style property is defined by a property name, followed by a colon, followed by a value. Style properties are separated by semicolons.
- Styles can be applied inline or at the document level, but these methods are error-prone and difficult to maintain. The most effective way to apply styles is with a style sheet.
- A master page is a template that holds content that you want to appear on all pages of your site. The master page also contains content placeholder areas where you can insert the content of each content page.
- To create a master page, select Website → Add New Item, and choose Master Page from the Add New Item dialog.
- To add content pages that use your master page, select Website → Add New Item, select Web Form from the Add New Item dialog, and be sure to check the "Select master page" checkbox. When you click Add, you'll be asked to choose which master page you want to use. Content pages have a `Page` directive that indicates their master page.
- You can nest one master page inside another. Create a new master page and delete all of the content except for the `Master` directive. Then, insert a `MasterPageFile` attribute that points to the primary master page. The new submaster page will appear inside the master page, with its own content placeholders.
- You can change the content of the master page at runtime by implementing a public property on the master page and adding a `MasterType` directive to the top of a content page. Once the content page has a reference to the class of the master page, the content page can programmatically interact with public properties of the master page.
- You can add simple `Button` controls that allow users to navigate from the current page to the button's target. The `Response.Redirect` method transfers the user directly to the new page without first posting back to the server.
- ASP.NET provides a number of predefined navigation controls that make it simple to help users move around your site.

- The site map is a data source that provides the information you need to implement a navigation tree, menu, or breadcrumbs on your site. It's an XML file that contains a series of nodes, each representing a page in your site. The IDE doesn't create this file, although it does provide a skeleton if you select Website → Add New Item and choose Site Map.
- After you've created the site map, you can add a `SiteMapDataSource` control to a page. The `SiteMapDataSource` control uses the file named *Web.sitemap* by default.
- Once you have a `SiteMapDataSource` in place, you can easily add a `TreeView` to your site. You set the data source for the `TreeView` to point to the `SiteMapDataSource`, and ASP.NET does the rest for you automatically. You can format the `TreeView` any way you like, from a set of predefined formats, or you can specify your own custom format.
- You can use a `Menu` control in the same way as the `TreeView` control. Simply add the Menu control to your page, point its data source to the `SiteMapDataSource`, and the IDE does the rest for you. You can change the format of the `Menu` control, or change the orientation from horizontal to vertical.
- You can access the nodes in your site map programmatically, using the `CurrentNode` and `ChildNodes` properties.
- Bread crumbs are a tool that indicates the current page and the preceding pages in the hierarchy. They're more concise and compact than a `TreeView` or a `Menu`. To use bread crumbs, place a `SiteMapPath` control on the page with the `SiteMapDataSource`. You don't need to specify the data source; it's done automatically. You can format the bread crumbs as you see fit.

This chapter is one of the first times in this book that you've seen sites that consist of multiple pages. Obviously, most sites in the real world consist of more than one page, and here you've taken your first steps toward more complex sites. You also learned in this chapter how you can make controls in content pages that affect the content of the master page. That's a great technique, but you'll notice that any content you change vanishes as soon as you navigate to another page. If you're going to have sites with multiple pages, you'll need some way of passing data from page to page, or between postbacks. That's called preserving *state*, and you'll learn several ways of doing it in the next chapter.

Quiz

1. What's the most effective way to apply styles on your web page?
2. If a style sheet has rules applying to the text of the whole page, but one specific paragraph has a different style rule applied, which takes precedence?
3. What two commands can you use to apply a style sheet to your page, and where do you place them?
4. What is the purpose of a master page?
5. How many different master pages can you associate with a particular content page?
6. When you are trying to change the content of a master page at runtime, what does the content page need to affect the master page?
7. What method could you use in a `Button Click` event handler to navigate to another page?
8. What file do you need for all the navigation controls to work? How is this file generated?
9. What control do you use to enable the navigation controls to access the file?
10. What do you have to do to connect the `SiteMapPath` control to a data source?

Exercises

Exercise 6-1. In this set of exercises, you'll create a web site for a travel agency called Ajax Travel. First, create a master page containing all of the elements that should appear on each page of the site, with the company name at the top, and a copyright notice at the bottom. In addition, Ajax Travel's portfolio of destinations is divided into two categories: Sun and Snow. The home page of the site should offer users a choice between these two categories. Each category should have its own heading, which appears in addition to the company heading. Also, each page in the Sun category should carry a message at the bottom reading "Ask about our honeymoon specials!" Each page in the Snow category should carry a message at the bottom reading, "Ask about our ski vacation packages!" To keep things simple, create just two content pages for each category: the Sun category should have one page for Bermuda, and one for Maui; the Snow category should have one page for Vail, Colorado, and one for St. Moritz, Switzerland.

The Maui page should look like Figure 6-30.

The Vail page should look like Figure 6-31.

Figure 6-30. *Your goal for the "Maui" page of Exercise 6-1.*

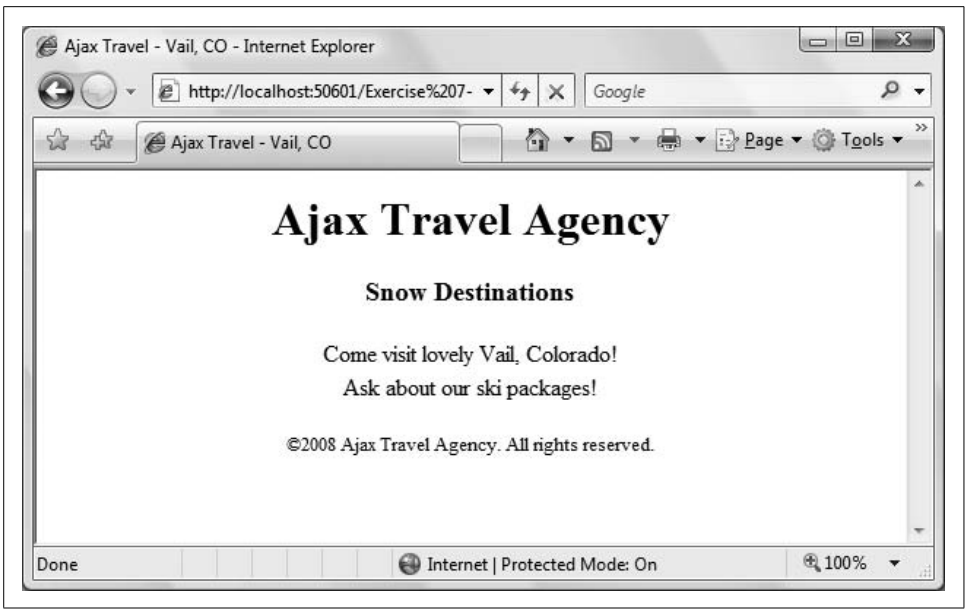


Figure 6-31. *Your goal for the "Vail" page of Exercise 6-1.*

Exercise 6-2. Add a control to the home page of the Ajax Travel site asking the user to enter his name. Update the master page of the site so that "Welcome, <name>!" is displayed at the top of the page. The home page should look like Figure 6-32.

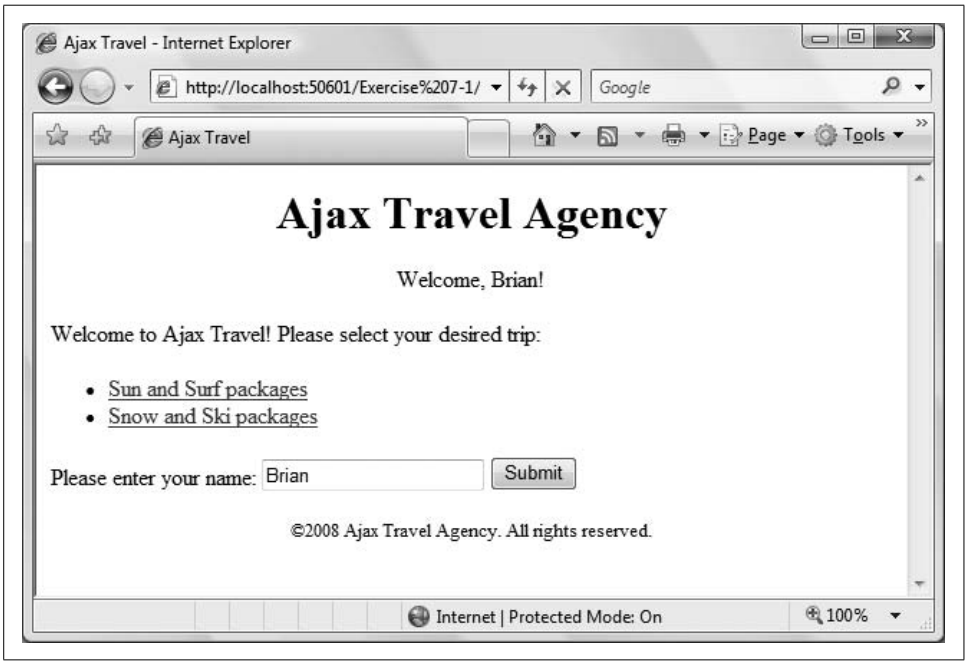


Figure 6-32. *Your goal for Exercise 6-2.*

Exercise 6-3. Remove the user greeting control from the master page. Implement a `TreeView` and a `Menu` for site navigation, both on the master page. Let users choose between the two types of navigation controls by using a control on the master page, as shown in Figure 6-33.

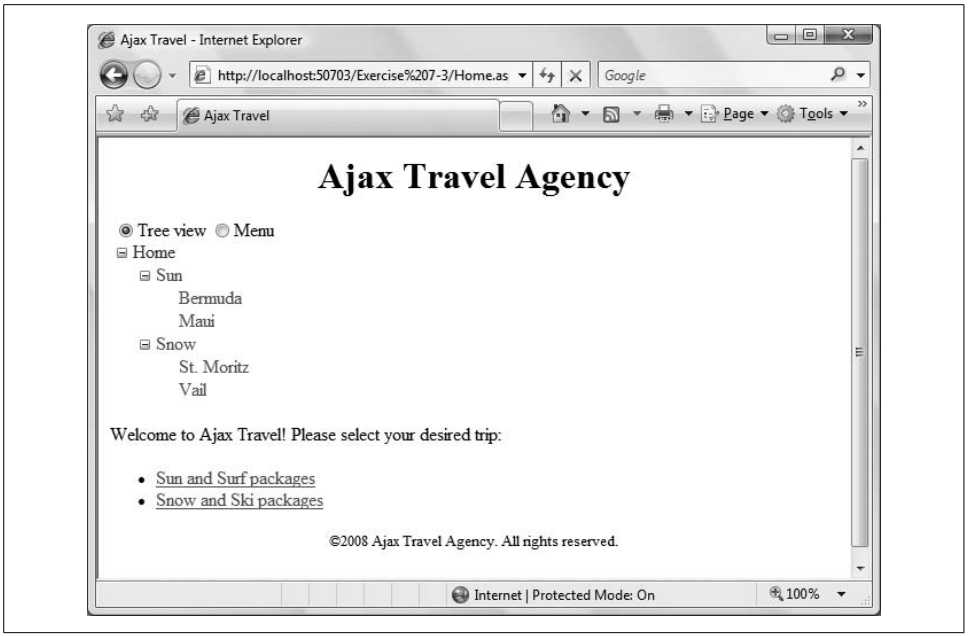


Figure 6-33. *Your goal for Exercise 6-3.*

Exercise 6-4. Add breadcrumbs to your site on the master page, as shown in Figure 6-34.

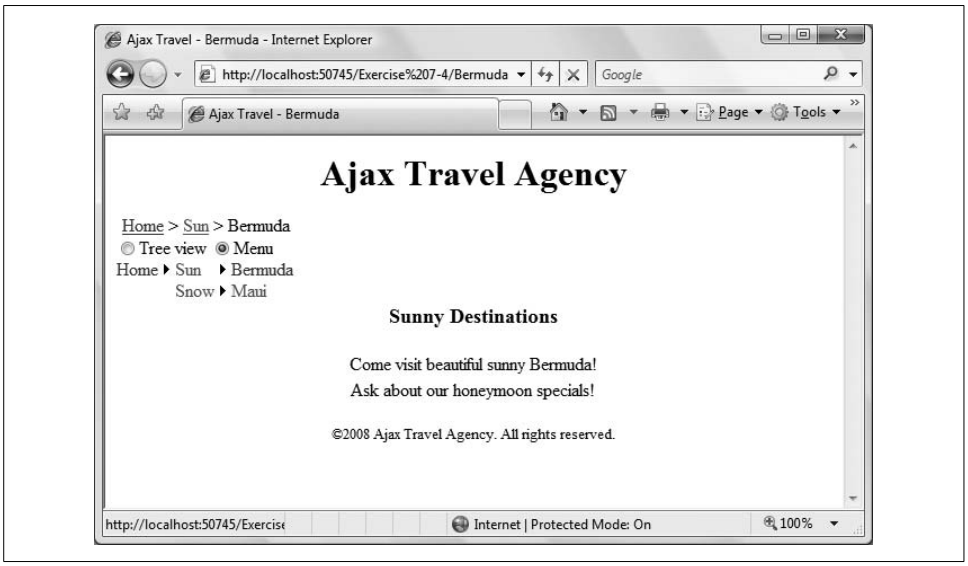


Figure 6-34. *Your goal for Exercise 6-4.*

CHAPTER 7

State and Life Cycle

Most of the web sites you have built in this book have been confined to a single page. In Chapter 6, you learned techniques to provide a single look and feel throughout the site.

When you created the examples in Chapter 6, you may have noticed that any data you entered on a page didn't stick around if you went to a different page and came back, even if you issued a postback on the page you were on. That's because the pages you've built up until now haven't had any way to preserve that kind of information, called *state*.

In each chapter up until now, you've learned about the different kinds of controls and how you use them. We're going to take a slightly different approach in this chapter, first taking you behind the scenes so you can understand what the page actually does when you click the Submit button. Next, you'll find out more about state, and how to hold onto it. By the time you've finished this chapter, you'll have built several sites that can retain state, no matter how much the user clicks around.

Page Life Cycle

A user sits at her browser and types in a URL. A web page appears with text, images, buttons, and so forth. She fills in a text box and clicks a button. New data appears in response. How does this work?

Before we begin, it is important to understand a little bit about the "architecture" of the World Wide Web. In the applications you've developed to this point, you've been able to do everything on one computer. But in order for these exercises to work, your single computer is standing in for three or four important pieces of the puzzle, as shown in Figure 7-1.

In the original model of the Web, a browser would send a request for a page, identified by a *Universal Resource Locater* (URL), and then some remote server would return that page. Information would be presented using HTML, a simple markup

language that the browser would display. Pages were imagined to be display-only, with interaction limited to clicking on hyperlinks to move from page to page. Each page was designed to be independent of all the others, and it didn't matter who was looking at a given page, or when.

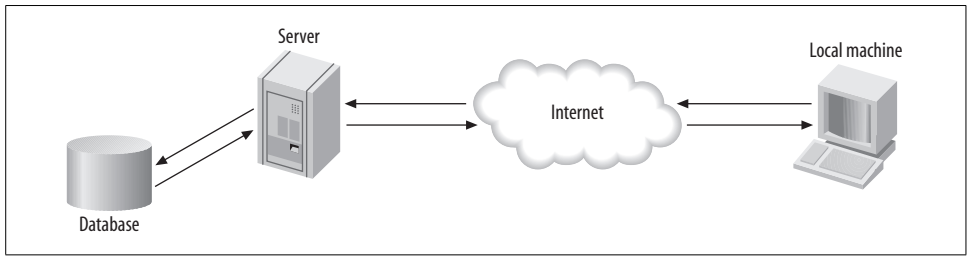


Figure 7-1. *When you enter a URL in your browser, your request crosses the Internet to the web server, which may access a database, and then returns the page to your local machine.*

In a traditional desktop application, on the other hand, you assume that a single user sits down, starts the application, uses it continuously for a while, and then closes the application. This period of time when the user is interacting with the application is called a *session*. If the user enters her name at some point, the application should hold onto that name for the duration of the session in case it needs to be retrieved at some later point. No matter where the user goes in the application, that name is assumed to be the same, and so the application may need to pass that name around to different methods within the code as needed. The user's name, along with any other changes she makes, is part of the *state* of the application, and the application needs to preserve that state for at least the duration of the session, and sometimes between sessions.

Desktop applications were always intended to preserve the application state; web pages were not. In fact, the Web was consciously and explicitly designed to be "stateless." This plan worked just fine for a while, but nowadays, web users expect web applications to behave like desktop applications. That is, if you enter something in a textbox, and then switch to work in a different application for a while, the content of the textbox is still there when you come back. That means web applications need a way to preserve state.

To create a session-based interactive application on top of the Web, ASP.NET uses pages that extend traditional HTML pages. These are stored on the web server and combine markup and code to produce HTML for the user's browser.

Some code runs on the server when the page is requested. Some code runs on the server in response to actions taken by the user (pressing a button, for example). Some code is embedded in the page and runs in the browser (AJAX and JavaScript client-side code).

To understand when different bits of code are run, and how the page that is sent to the browser is assembled, you need to understand the *life cycle* of the ASP.NET page.

When an ASP.NET page is requested from the server, the server loads the page into server memory, processes the page, sends the page to the user, and then unloads it from memory. From the beginning of the life cycle to the end, the goal is to render appropriate HTML to the requesting browser. At each step, methods and events are available that allow you to override the default behavior or add your own programmatic enhancements.

In ASP.NET, the `Page` class handles the responses. When a browser requests a page, an instance of the `Page` class is created, and that instance creates a hierarchical tree of all the controls on the page: the *control tree*.

To see this, you'll create a simple web site called *LifeCycle*. This site will consist of a single page containing a single `Button` control. Clicking on the `Button` will do nothing but cause the page to post back to the server.

In Source view, open *Default.aspx*, and then drag a `Button` control onto the page between the two `<div>` tags.

Add the `Trace` attribute to the `Page` directive at the top of the file and set its value to `true`, as shown in Figure 7-2.

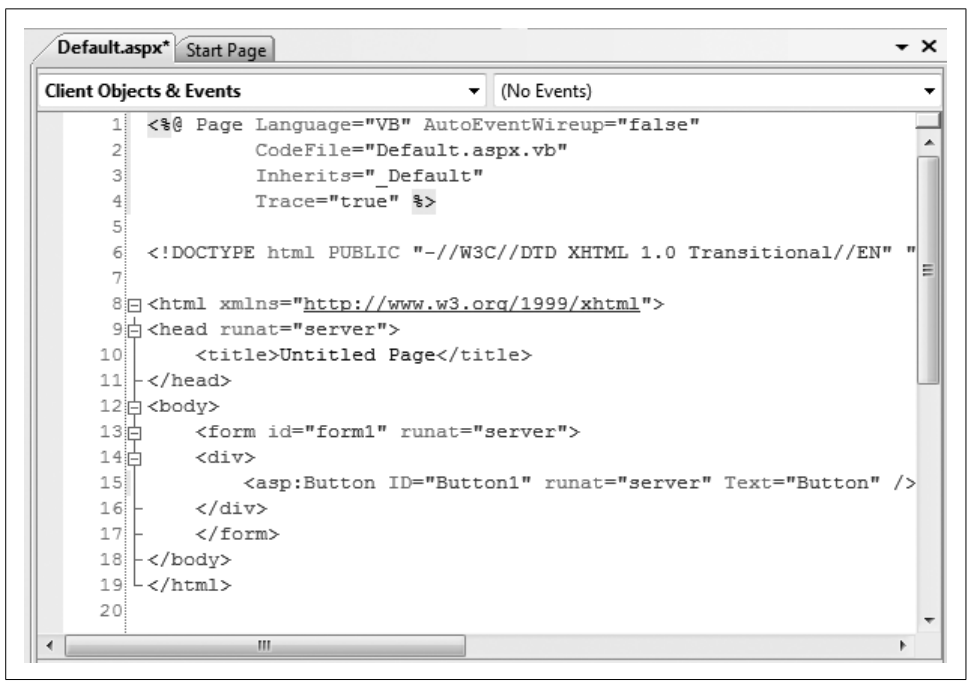


Figure 7-2. *The LifeCycle Default page in Source view shows the single control on the page and the Trace attribute within the Page directive.*

We haven't yet formally introduced the `Trace` control, but you need it for this exercise. You can see the control tree for any page by adding `Trace="true"` to the `Page` directive. We'll cover tracing in detail in Chapter 8.

Run the page. You will see the single button at the top, followed by a ton of information at the bottom. For now, just slide down to the section labeled Control Tree, as shown in Figure 7-3.

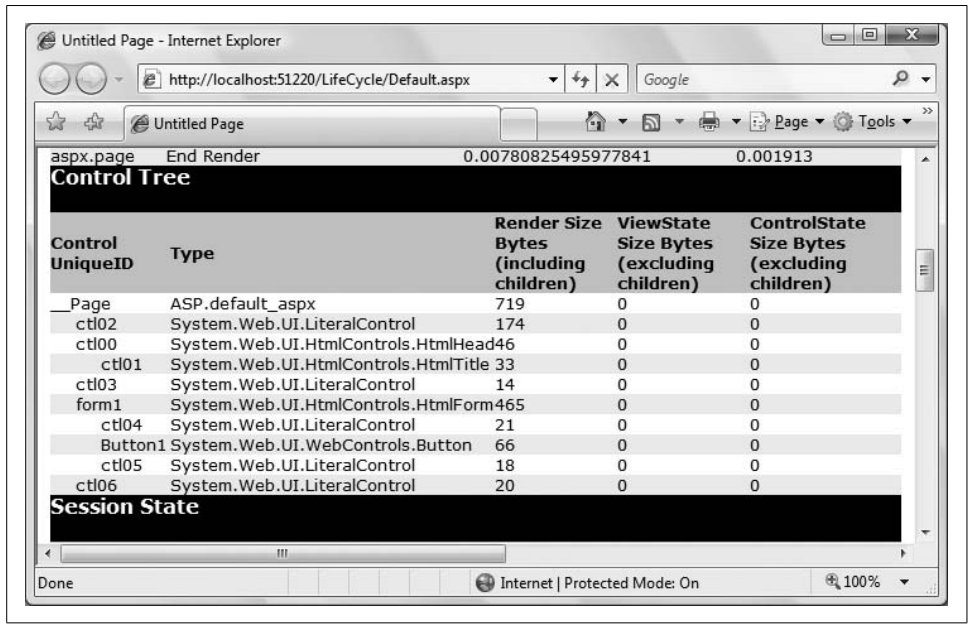


Figure 7-3. *The page Trace contains a lot of information about the page, but for now, you're only interested in the Control Tree section, which shows all the controls on the current page.*

The `Page` itself is at the root of the tree. All the named controls are included in the tree, referenced by control ID. In our simple example, there are only two named controls: `form1` and `Button1`.

Static text, including whitespace, newlines, and HTML tags, are represented in the tree as `LiteralControls`. The order of controls in the tree is strictly hierarchical. Within a given level of the hierarchy, the controls are in the order in which they appear in the markup file.

Web components, including the `Page`, go through their entire life cycle every time the page is loaded. Events fire first on the `Page`, then recursively on every object in the control tree. Just to emphasize the point: every time the page is posted back to the server, it goes away, and an entirely new page, with an entirely new set of controls, is created by the server and sent to the browser. This page and these controls will be nearly identical to those on the previous page.

There are two slightly different sequences in the life cycle: one for the first time a page is loaded, and a second when the page reloads itself in a postback. The life cycle is shown schematically in Figure 7-4.

During the first page load, the life cycle consists of the following steps:

1. A request for the page is made from a browser to the web server. The ASP.NET Framework first determines whether the page already exists in a cache (a section of memory specifically reserved for recently used items). If so, the page is retrieved and returned to the browser and we are done. If not, then the actual page life cycle starts at this point.
2. During the Start phase, the postback mode is determined. If the page was requested by a different page, then it was not a postback. If the page was returned to the server for processing and redisplay, then it is a postback. The `IsPostBack` and `PreviousPage` properties are set accordingly. The Request and Response properties of the page along with a number of other properties are also set.
3. The Page Initialization phase contains two events often handled by your code: `PreInit` and `Init`. If you do not handle these explicitly yourself, ASP.NET will perform the default behavior on your behalf. During the `PreInit` event, the target device is determined before the page is initialized, the master page is set, the control tree is built, and the controls are assigned unique IDs, which are made available to your code. Personalization and themes are loaded and applied to the page in this step (these are discussed in Chapter 10). `PreInit` is the first event in the life cycle that can be trapped and handled. That is, this is the first event that you can write your own code for, to change the default behavior of initializing the page.
4. During the `Init` event, control properties are read or initialized. If this is a postback, it is important to realize that the controls won't reflect any changes to the page made before the postback—that happens in the `PreRender` phase. They *will* contain values specified in the markup file.
5. During the `Load` event, all the control properties are set. View state information is available, and controls in the page's control hierarchy can be accessed. The load phase is routinely modified in a `Page_Load` method.
6. During the Validation phase, the `Validate` method is called on all the validation controls on the page. The `IsValid` property is set for all those controls and for the page as a whole.
7. During the Rendering phase, personalization, control, and view state is saved. Each control on the page is called in succession to render itself to the browser, that is, to compose itself into HTML that is included in the page's `Response` property. It is very common to handle the `PreRender` event with a `Page_PreRender` method, typically when you must take some action based on the final value of some other control on the page. During the `Render` event, the HTML is actually

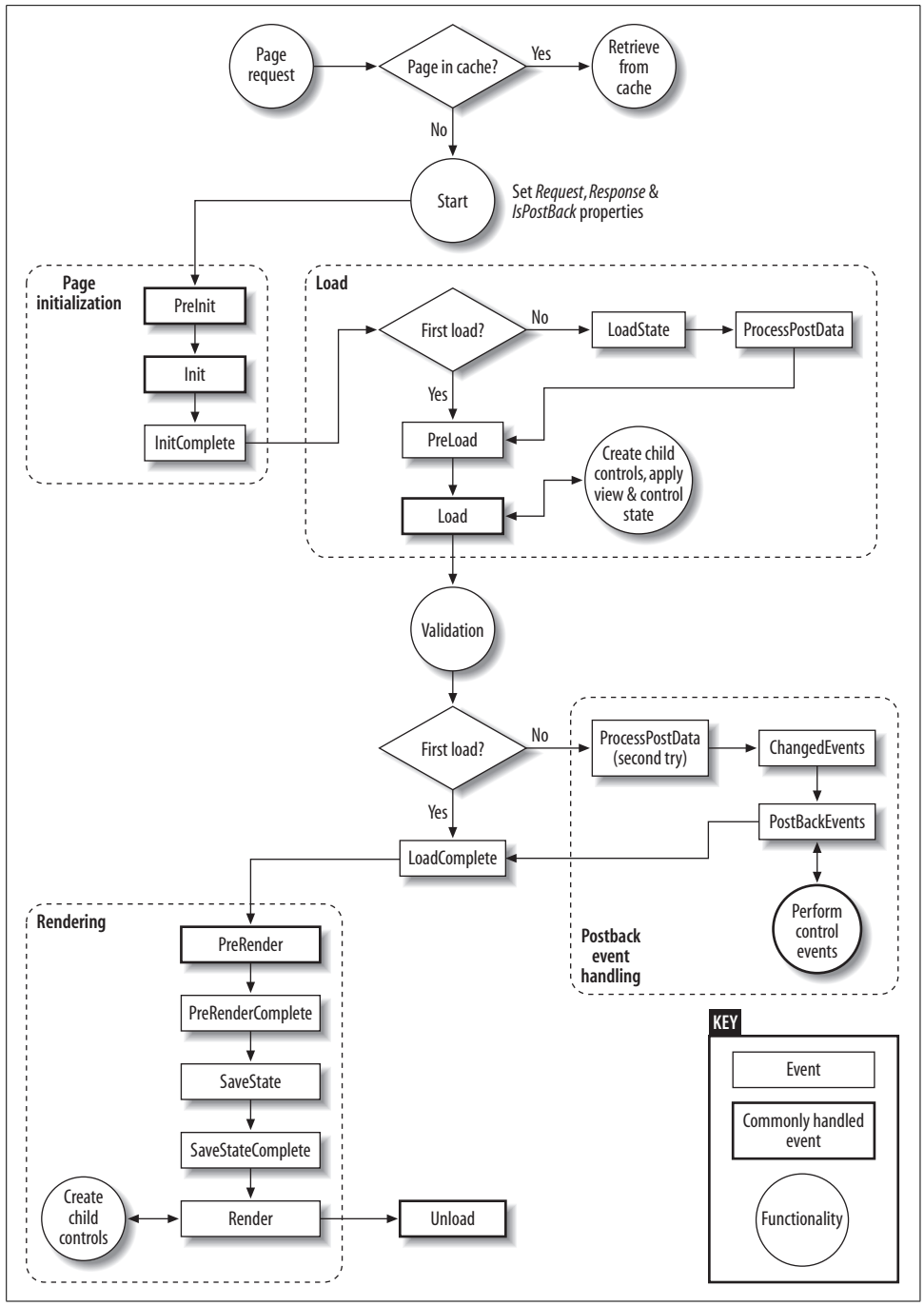


Figure 7-4. *Schematic of ASP.NET page life cycle. Events are in rectangles, with commonly handled events in bold rectangles. Areas of functionality are indicated in circles. The dashed round-cornered rectangles delineate the major phases of the cycle.*

generated and sent to the requesting page, although this event is rarely handled unless you are developing custom controls.

8. `Unload` is the last event of the life cycle. It gives you an opportunity to do any final cleanup (i.e., closing open files and releasing references to expensive resources, such as database connections).

During postback, the life cycle is the same as during the first load, except for the following:

1. During the `Load` phase, after initialization is complete, the view state and the control state are loaded and applied as necessary.
2. After the Validation phase completes, postback data is processed. Control event handlers are now executed. This is important: control event handlers, such as a Button `Click`, are not called until *after* the Page Initialization and Load events are handled. This is important because it is often critical in which order code in the various event handlers is executed.

You can easily see the order in which events are fired on a page by turning on tracing for the page as you did above (setting `Trace` to `true` in the `Page` directive). As shown in Figure 7-5, the Trace Information section of the sample lists all the page events along with the number of seconds it took for that event to run from the start of the life cycle and from the previous event.

If you click the button to cause a postback, the trace information will include the additional events indicated previously in Figure 7-4, including `ProcessPostData Second Try`, `ChangedEvents`, and `PostBackEvent`. Often, when you're trying to track down why your page is behaving a certain way or why some of your code does not seem to work as you would expect, looking at the life cycle behavior can be illuminating.

State

State, in the case of a web page, is the current value of all the controls and variables for the current user, in the current session. The Web is inherently a *stateless* environment, which means that each time a page is posted to the server and then sent back to the browser, the page is created again from scratch. Unless the state of all the controls is explicitly preserved before the page is posted, the state is lost and all of the controls will be created with default values. One of the great strengths of ASP.NET is that it automatically maintains state for server controls—both HTML and ASP.NET—so you do not have to write any code to accomplish this. This section will explore how this is done and how you can make use of the ASP.NET state management capabilities.

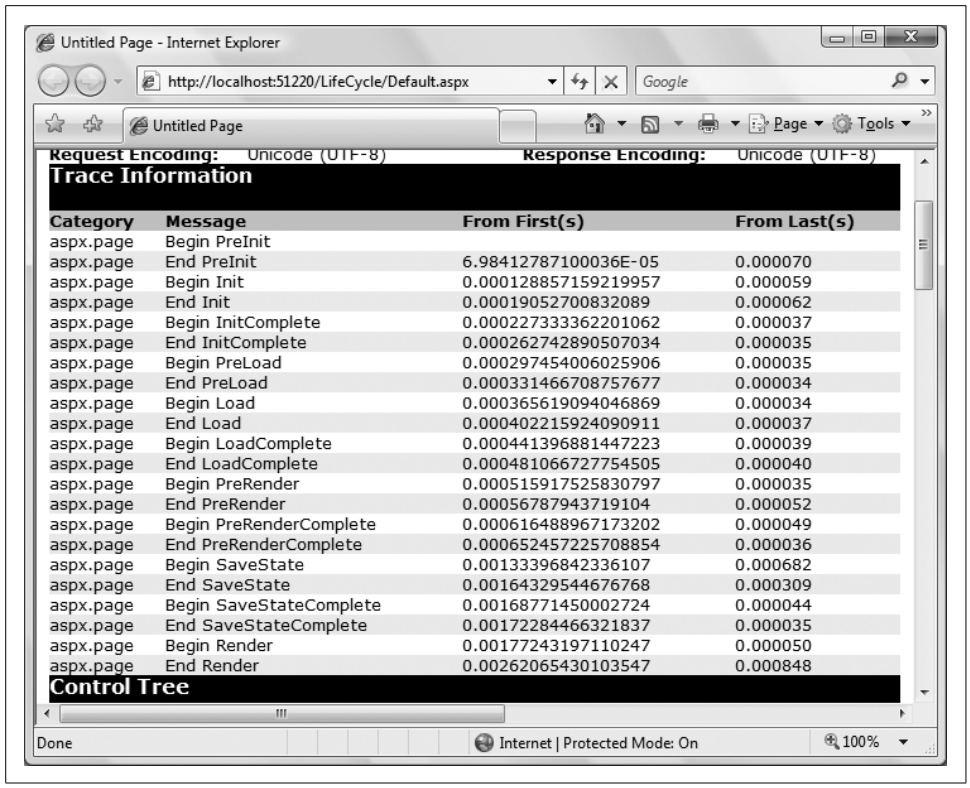

Untitled Page - Internet Explorer
http://localhost:51220/LifeCycle/Default.aspx
Google
Untitled Page
Page
Tools
Request Encoding: Unicode (UTF-8)
Response Encoding: Unicode (UTF-8)
Trace Information
Category Message From First(s) From Last(s)
aspx.page Begin PreInit
aspx.page End PreInit 6.98412787100036E-05 0.000070
aspx.page Begin Init 0.000128857159219957 0.000059
aspx.page End Init 0.00019052700832089 0.000062
aspx.page Begin InitComplete 0.000227333362201062 0.000037
aspx.page End InitComplete 0.000262742890507034 0.000035
aspx.page Begin PreLoad 0.000297454006025906 0.000035
aspx.page End PreLoad 0.000331466708757677 0.000034
aspx.page Begin Load 0.000365619094046869 0.000034
aspx.page End Load 0.000402215924090911 0.000037
aspx.page Begin LoadComplete 0.000441396881447223 0.000039
aspx.page End LoadComplete 0.000481066727754505 0.000040
aspx.page Begin PreRender 0.000515917525830797 0.000035
aspx.page End PreRender 0.00056787943719104 0.000052
aspx.page Begin PreRenderComplete 0.000616488967173202 0.000049
aspx.page End PreRenderComplete 0.000652457225708854 0.000036
aspx.page Begin SaveState 0.00133396842336107 0.000682
aspx.page End SaveState 0.00164329544676768 0.000309
aspx.page Begin SaveStateComplete 0.00168771450002724 0.000044
aspx.page End SaveStateComplete 0.00172284466321837 0.000035
aspx.page Begin Render 0.00177243197110247 0.000050
aspx.page End Render 0.00262065430103547 0.000848
Control Tree
Done
Internet | Protected Mode: On
100%

Figure 7-5. *The Trace Information section of the page Trace shows all the page events.*

ASP.NET manages four types of state:

Control state
: Used to provide features such as paging and sorting of `GridView` controls. Control state cannot be modified, accessed directly, or disabled.

View state
: The state of all the controls on the page. View state only lasts for that one page display, and is updated every time the page is redrawn. It can be disabled for specific controls, the page, or the entire web site.

Session state
: Data specifically saved across page posts, for use by all the pages in a web application.

Application state
: Data available to all the users of a web application, even across multiple sessions.

Table 7-1 compares the kinds of state management (other than Control state, which is not accessible to the developer).

Table 7-1. *Comparison of types of state*

Feature	View state	Session state	Application state
Uses server resources	No	Yes	Yes
Uses bandwidth	Yes	No	No
Times out	No	Yes	No
Security exposure	Yes	Depends	No
Optimized for nonprimitive types	No	Yes	Yes
Available for arbitrary data	Yes	Yes	Yes
Programmatically accessible	Yes	Yes	Yes
Scope	Page	Session	Application
Survives restart	Yes	Depends on configuration	No

View State

The *view state* is the state of the page and all its controls. View state is automatically maintained across posts by the ASP.NET Framework. When a page is posted to the server, the view state is read. Just before the page is sent back to the browser, the view state is restored.

The view state is saved in a hidden field on the page. Because the view state is maintained via a form field, this technique works with all browsers. The information saved in the hidden field is Base64 encoded, but not encrypted. As such, any information stored in view state is not immune to prying eyes.

If there is no need to maintain the view state for a given page, you can boost performance by disabling view state for that page. For example, if the page does not post back to itself or if the only control on a page that might need to have its state maintained is populated from a database with every round trip to the server, then there will be no need to maintain the view state for that page. To disable view state for a page, add the `EnableViewState` attribute with a value of `false` to the `Page` directive:

```
<%@ Page Language="VB"  EnableViewState="false" %>
```

The default value for `EnableViewState` is `true`.

You can also disable the view state for an entire application by setting the `EnableViewState` property to `false` in the `<pages>` section of the *web.config* configuration file.

You can even maintain or disable the view state for individual controls. This is done with the `Control.EnableViewState` property, which is a Boolean value with a default of `true`. Disabling view state for a control, just as for the page, will improve performance. This would be appropriate, for example, in a situation where a `GridView` is populated from a database every time the page is loaded. In this case, the contents of the control would be overridden by the database query, so there is no point in maintaining view state for that control. If the `GridView` in question were named `gv`, the following line of code would disable its view state:

```
gv.EnableViewState = false;
```

The simple controls, `TextBox`, `RadioButton`, `CheckBoxList`, and `RadioButtonList`, ignore the `EnableViewState` properties, as the current value is always saved in view state and posted back to the server anyway. Complex controls such as the `GridView` do respect the `EnableViewState` property. In cases where a `DataSource` control is used as the data source for a `GridView`, the data is gathered fresh with every page post. In this case, it is more performant to disable view state.

There are some situations where view state is not the best place to store data. If you need to store a large amount of data, view state is not an efficient mechanism because the data is transferred back and forth to the server with every page post. If you have security concerns about the data, and the data is not being displayed on the page, then including it in view state increases the security exposure. Finally, view state is optimized only for strings, integers, Booleans, arrays, ArrayLists, and hashtables. Other .NET types may be saved in view state, but will result in degraded performance and a larger view state footprint.

In some of these instances, session state might be a better alternative; on the other hand, view state does not consume any server resources and does not time out as session state does.

To see view state in action, create a new web site called ViewState.

This web site will have a `TextBox`, a `Label`, and a `GridView` control. It will also have a `Button` control to force a postback to the server. To better demonstrate view state, you will not use a `DataSource` control to populate the `GridView`; rather, you'll use ADO.NET code in the code-behind to gather the data the first time the page loads (don't worry, we'll show you what to do).

The finished Design view of the page will look similar to Figure 7-6.

While in Design view, drag a `Button` control and a `TextBox` control onto the page. Change the `Text` property of the Button to `Post` to indicate that it will post back to the server.

Press the Enter key several times, and then drag a `Label` control onto the page.

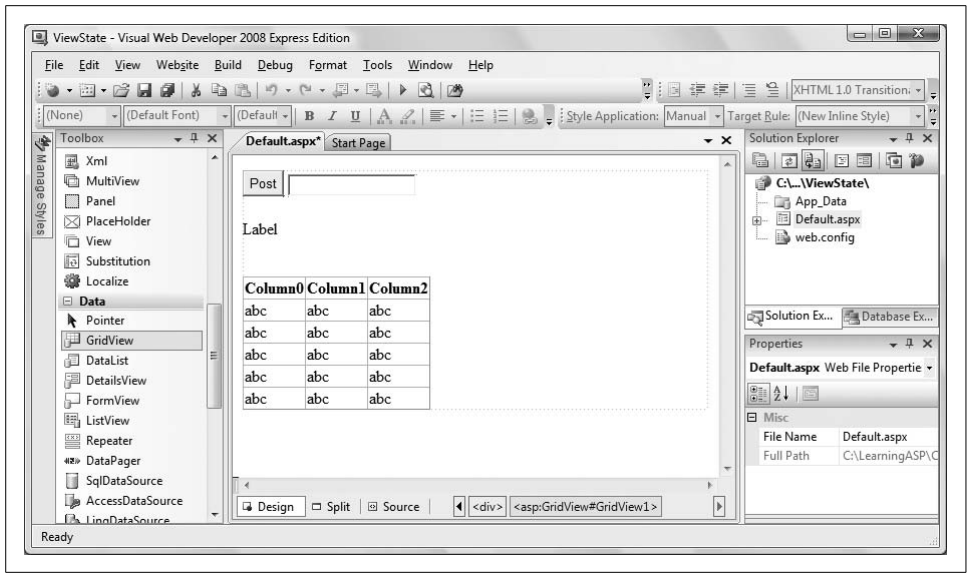


Figure 7-6. *You'll create the ViewState page for this example so that it looks like this in Design view.*

Press the Enter key a few more times. Now, drag a `GridView` control from the Data section of the Toolbox onto the page.

The `GridView` will be populated from the AdventureWorks database. In preparation for this, add a connection string to the *web.config* file by double-clicking on that file in the Solution Explorer to open the file in the editing window.

Replace this single line (if it exists):

```
<appSettings />
```

with the lines that follow. If the above line of code is not in the file, add the text below after the opening `<configuration>` element anyway (the quoted value should not wrap in your code, as it does on this printed page):

```
<appSettings>
    <add key="AdventureWorks"
        value="Data Source=<server name>;
            Initial Catalog=AdventureWorks;
            Integrated Security=True;" />
</appSettings>
```

Replace the Data Source with the appropriate SQL Server name for your machine. If you are using SQL Express, the Data Source will be something like (note the single period before the back slash):

```
Data Source=.\SqlExpress;
```

Open the code-behind for the page by double-clicking on *Default.aspx.vb* in the Solution Explorer. Create an event handler for the Page Load event by selecting (`Page Events`) in the Class Name drop-down menu at the top left of the editing window, and in the Method Name drop-down at the top right of the editing window, select the `Load` event. This will insert an empty code skeleton for the `Page_Load` event handler.

Type the highlighted code inside the `Page_Load` method from Example 7-1. (Note the two `Imports` statements that must appear before the `Partial Class` line.)

Example 7-1. *The Default.aspx.vb for the ViewState web site has a simple Page_Load method that calls a more elaborate helper method*

```
Imports System.Data
Imports System.Data.SqlClient

Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles Me.Load
        If Not IsPostBack Then
            Label1.Text = "Hello World"
            PopulateGrid( )
        End If
    End Sub

    Private Sub PopulateGrid( )
        Dim connectionString As String = _
                ConfigurationManager.AppSettings("AdventureWorks")
        Dim connection As SqlConnection = _
            New SqlConnection(connectionString)
        Dim queryString As String = _
            "select top 1000 AddressLine1, AddressLine2, City, " + _
                "StateProvinceID, PostalCode from Person.Address"
        Dim ds As DataSet = New DataSet( )
        Try
            Dim dataAdapter As SqlDataAdapter = _
               New SqlDataAdapter(queryString, connection)
            dataAdapter.Fill(ds, "Addresses")
            GridView1.DataSource = ds.Tables("Addresses")
            GridView1.DataBind( )
        Catch ex As Exception
            '' Handle exception
        Finally
            connection.Close( )
        End Try
    End Sub

End Class
```

Also from Example 7-1, add the helper method called `PopulateGrid`, which actually does the work of gathering and binding the data for the `GridView`. (Again, as with the `Trace` control earlier, it is not important in this example for you to understand exactly how `PopulateGrid` works.) In order for `PopulateGrid` to build properly, you must include the two `Imports` statements at the top of the code-behind file in Example 7-1.

VB CHEAT SHEET

Helper Methods

When your event handler has a lot of code that you might want to use again elsewhere, it's a good idea to separate that code out into another method. You can run that method from various other points in your code, which is known as *calling* the method. To create a helper method, you create a new `Sub` in the code-behind file. Be careful to place it before the `End Class` statement and give it a name. The following example shows what this looks like:

```
Private Sub PopulateGrid( )
' Your code goes here.
End Sub
```

To call the method, you simply type the name the function, with the parentheses, at the point in the code where you want the method to execute, as in:

```
PopulateGrid( )
```

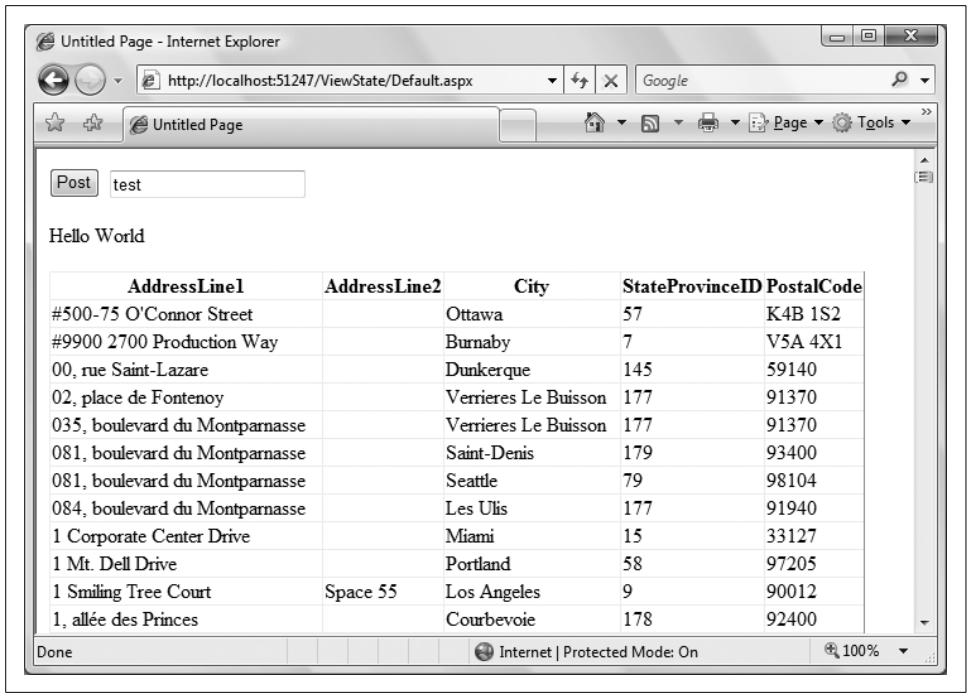
The execution of the application jumps to the beginning of the method, and then executes the method. When the execution reaches the `End Sub` statement at the end of the helper method, it returns to the line in the event handler where it left from and continues from there.

If you need to pass values (called parameters) from the calling method into the helper method, you would include them in the parentheses of the method call. This example doesn't pass any parameters.

Run the application. Enter some text in the `TextBox` and observe the result. Your screen will appear similar to Figure 7-7.

If you run the page and there is no data visible in the grid after clicking the Post button, you've probably disabled view state for the page (as you are instructed to do below). Make sure the `EnableViewState` attribute is set to `true` in the Page directive at the top of the markup file:

```
<%@ Page Language="VB" AutoEventWireup="false"
CodeFile="Default.aspx.vb" EnableViewState="true"
Inherits="_Default" %>
```



AddressLine1	AddressLine2	City	StateProvinceID	PostalCode
#500-75 O'Connor Street		Ottawa	57	K4B 1S2
#9900 2700 Production Way		Burnaby	7	V5A 4X1
00, rue Saint-Lazare		Dunkerque	145	59140
02, place de Fontenoy		Verrieres Le Buisson	177	91370
035, boulevard du Montparnasse		Verrieres Le Buisson	177	91370
081, boulevard du Montparnasse		Saint-Denis	179	93400
081, boulevard du Montparnasse		Seattle	79	98104
084, boulevard du Montparnasse		Les Ulis	177	91940
1 Corporate Center Drive		Miami	15	33127
1 Mt. Dell Drive		Portland	58	97205
1 Smiling Tree Court	Space 55	Los Angeles	9	90012
1, allée des Princes		Courbevoie	178	92400

Figure 7-7. *After you've entered some text in the TextBox, the ViewState page will look like this. This is the expected result with View state enabled.*

Click on the Post button. The page will be posted back to the server, as indicated by the progress indicator in the status line at the bottom of the browser. However, nothing on the page will change.

The contents of the `TextBox` are preserved by the built-in view state capability of ASP. NET. You have written no code anywhere to do this, it just happens.

Looking at the `Page_Load` event handler, you can see that the `Text` property of the `Label` control is set when the page is first loaded, but not on subsequent post back. Ditto for the `GridView`—it is populated only the first time the page is loaded. Again, ASP.NET view state is taking care of preserving the data between postbacks.

Now, watch what happens when you disable view state for the page. Open *Default.aspx* in Source view. Add the `EnableViewState` attribute to the `Page` directive at the top of the file, and set its value to `false`, as shown highlighted in the following code snippet:

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    EnableViewState="false" Inherits="_Default" %>
```

Run the page again and enter some text in the TextBox. It initially looks the same as Figure 7-7. Click the Post button to post the page back to the server.

The page is very different, as shown in Figure 7-8.

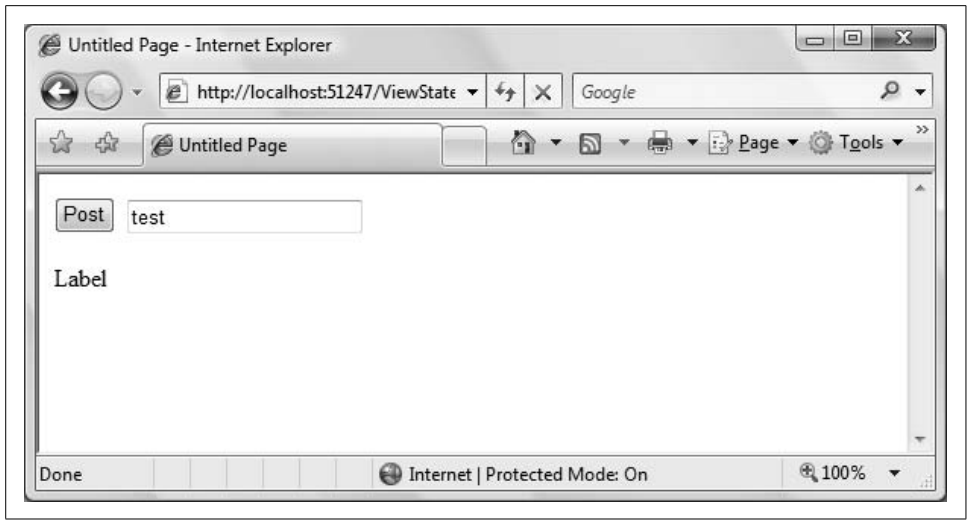


Figure 7-8. *When you post the page back with View state disabled, the results are very different.*

The `TextBox` still shows the current value. As mentioned in the note above, simple controls such as `TextBox` ignore the `EnableViewState` property and always preserve view state. However, the `Label` and `GridView` controls do respect that property. Because they are only populated the first time the page is loaded, they no longer display the current data after the page is posted back to the server. The `Label` reverts to its default `Text` property, and the `GridView` is not even rendered if there is no data bound to it.

In addition to preserving the values of controls, view state is very handy for something else—you can programmatically stash your own stuff in a data structure known as the *state bag*, using the `ViewState` keyword. The state bag stores data as attribute/value pairs in a dictionary. The attribute is a string, which is the name of the object stored as the value. The types of objects that can be stored efficiently are the primitive data types, including integers, strings, bytes, and Booleans. You can also store arrays of primitive types. Complex types, such as DataSets and custom objects can be stored as well, but performance will suffer.

The next example demonstrates stashing values in the state bag and then retrieving them. In this example, a counter keeps track of the number of times a button on a page has been clicked. As long as the page is current, the count will be correct. If you move to another page and then return to this page, the count will start over because view state will be reinitialized for the new page.

Create a new web site called *StateBag*. On the Default page, drag on a `Label`, name it `lblCounter`, and then add two buttons. Set the `Text` property of one button to `Increment Counter`, and the `Text` property of the other button to `Navigate`.

VB CHEAT SHEET

Arrays and Dictionaries

You've seen variables in previous chapters, and you've seen collections, like the collection of `ListItems` in a `RadioButtonList`. An array is simply another type of collection, in which you can store a bunch of objects in a single variable, provided they're all of the same type. You declare an array similar to the way you would create a variable:

```
Dim myArray(5) As Integer
```

This code creates an array called `myArray`, which can hold integers. Specifically, the `(5)` states that the array can hold six integers. Why not five as the code suggests? Because just as with controls, array indexes begin at zero. To access the third integer in the array, you'd just use this syntax:

```
myArray(2)
```

Use curly braces if you want to initialize the array when you create it:

```
Dim myArray( ) As Integer = {42, 36, 128, 53, 7, 85}
```

Notice you don't have to specify the length of the array in this case; the compiler will automatically set it to a length of six.

A *two-dimensional* array is like an array of arrays-each entry of the array is itself an array, which simulates rows and columns. So, to access the fifth entry in the third array, you use this syntax:

```
myTwoDimensionalArray(2,4)
```

Arrays are particularly useful with `ForEach` loops, like this:

```
ForEach i In myArray
  i = i + 1
Next i
```

That little loop increments each element in the array by 1, and you don't need to know what values are in the array, or how many there are. One drawback to arrays is that if you want to retrieve a value from an array, you either have to know the index of the value you want, or else loop through the array until you find it. The *dictionary* is a specific kind of array (also known as a *Hashtable*) that solves this problem by associating each value with a *key*, instead of an index. For example, you could have a dictionary of U.S. states, using their abbreviations as the key values. Then, to retrieve a state's name, you'd just have to know the abbreviation (the key). The following code snippet creates and partially populates a `Hashtable` to hold the states:

```
Dim States as New Hashtable( )
States.Add("CA", "California")
States.Add("MA", "Massachusetts")
States.Add("PA", "Pennsylvania")
```

To retrieve the name of the dictionary entry with the key value of MA, you would use the following code:

```
Dim strStateName as string = States("MA").ToString( )
```

In the case of the state bag, the attribute names are the keys, and the values of those attributes are stored as the value part of each dictionary pair.

Add another page to the web site by clicking on Website → Add New Item…. Call this new page *AnotherPage.aspx*. Be sure the "Place code in separate file" checkbox is checked. On that page, add a `Button` control, with its `Text` property set to `Home`.

While still in *AnotherPage.aspx*, switch to Design view, and double-click the Home button to open the `Click` event handler. Add the highlighted code below:

```
Protected Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button1.Click
    Response.Redirect("Default.aspx")
End Sub
```

This button will now navigate back to the Default page.

Open *Default.aspx* in Design view. Double-click the Navigate button to open an event handler for that button, and add the highlighted code in Example 7-2 to the `Click` event handler.

Right-click on *Default.aspx* in the Solution Explorer and set it as the Start page. Run the web site and verify you can navigate back and forth between the two pages.

Stop the web site, then open the code-behind for the Default page, *Default.aspx.vb*. It already has an event handler for the Navigate button. There is no code necessary for the Increment Counter button.

Add an event handler for the `Load` event for the Default page. Next, add the highlighted code from Example 7-2 to the `Page_Load` method.

Example 7-2. *The Default.aspx.vb for the StateBag web site has a Page_Load method that increments the counter each time the page is loaded*

```
Partial Class _Default
    Inherits System.Web.UI.Page

Protected Sub Button2_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Button2.Click
    Response.Redirect("AnotherPage.aspx")
End Sub

Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    ViewState("Counter") += 1
    lblCounter.Text = ViewState("Counter").ToString( )
End Sub
End Class
```

Run the site. It will open with something like Figure 7-9. The counter will be initialized to 1.

Click the Increment Counter button. You will see the counter increment in the label. Navigate to the other page and back, however, and the counter will be reset back to 1. Just as in the previous example, the view state is retained through postbacks for controls on the same page, but once you transfer to a different page, that state is abandoned. This is called *scope*—the `ViewState` data is scoped to the page.

Figure 7-9. *When you open the StateBag page, the counter initializes to 1.*

VB CHEAT SHEET

+= Operator

The += operator is VB shorthand for adding the specified amount, and assigning the result to the original variable—in this case, incrementing by 1. The following two statements are equivalent:

```
myVariable = myVariable + 1
myVariable += 1
```

There are also -=, *=, and /= operators. In addition, there is an &= operator specifically for concatenating strings, although += works for strings as well.

In the `Page_Load` method in Example 7-2, examine the following line:

```
ViewState("Counter") += 1
```

This line creates a `ViewState` dictionary object if it does not already exist, creates a key in the dictionary called `Counter`, and associates the value of 1 with `Counter`. When you click the Increment Counter button, the page posts back, which means that the `Load` event is raised, and the method is called again. The `Counter` item already exists in the dictionary, so its value is increased by 1. Note that if the page had a dozen buttons, clicking any of them would increment the counter, as all it takes is a postback to run the code in `Page_Load`.

It is interesting to note that if you simply refresh the browser, rather than click a button on the page, the counter will not be incremented.

For this example, that works fine. However, in the general case, it is usually best to first test to see if the object in view state exists before trying to use it.

Consider the case where a string array is put into the state bag in `Page_Load` with the following lines of code:

```
Dim strArray( ) As String = New String( ) {"a", "b", "c"}
ViewState("TestArray") = strArray
```

Then, somewhere else in your code, maybe a Button `Click` event handler, for example, you want to retrieve the contents of that `ViewState` object and do something with it. You must first verify that the object exists before trying to use it, because if for some reason it does not exist, it will throw an exception and crash your program. You can verify the object's existence with an `If...Then` block that tests to see if the `ViewState` object is not `Nothing`, as in the following code snippet:

```
If ViewState("TestArray") IsNot Nothing Then
    Dim strNewArray( ) As String
    strNewArray = CType(ViewState("TestArray"), String( ))
End If
```

Then, once you are sure the `ViewState` object exists, you retrieve it by using the `CType` function to explicitly convert the object to the desired type. This is necessary no matter what type is stashed into the `ViewState` object because regardless of what type of object is stashed in the bag, what comes out is of type `Object` unless you convert it back.

Session State

When you connect to an ASP.NET web site, you create a *session*. The session imposes state on the otherwise stateless Web and allows the web site to recognize that subsequent page requests are from the same browser that started the session. This allows the web site to maintain state across pages until you explicitly end the session by closing the browser, or until the session times out. (The default timeout is 20 minutes, which you can change by editing the *web.config* file.)

The scope of a session assumes a single user making many different page requests. The session is not lost until the timeout period goes by with no requests from the browser. If the user goes to lunch and does not click on anything for more than 20 minutes (assuming the default timeout period), the session will terminate. On the other hand, if she clicks once every 19 minutes, the session will be maintained indefinitely.

While an application is running, there may be many sessions, essentially one for each user interacting with the web site, as indicated in Figure 7-10.

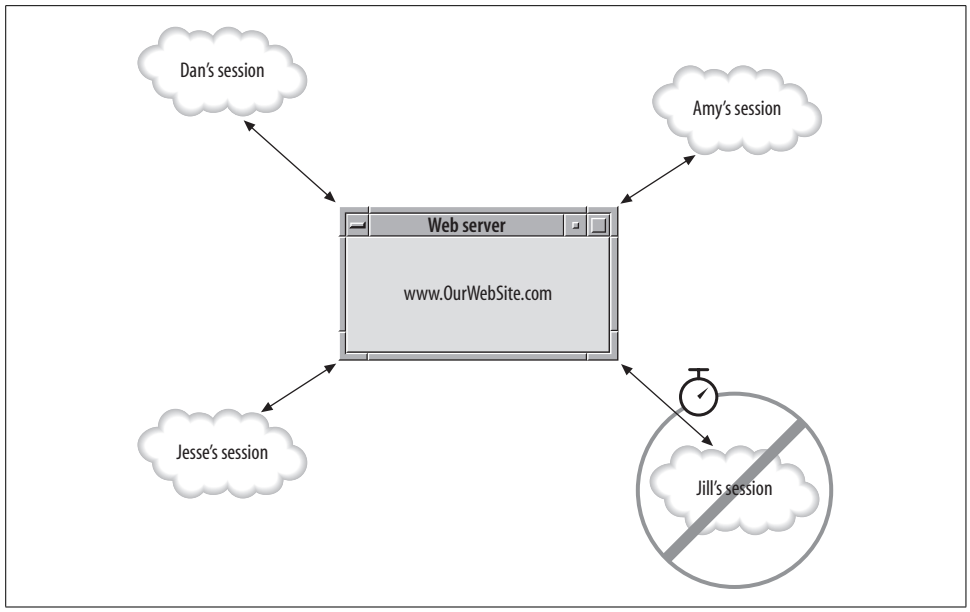


Figure 7-10. *A web site can have many simultaneous sessions, one for each user who has not timed out.*

ASP.NET provides session state with the following features:

- Works with browsers that have cookies either enabled or disabled.
- Identifies if a request is part of an existing session.
- Stores session-scoped data for use across multiple requests.
- Raises session events such as `Session_Start`, which you can handle in application code.
- Automatically releases session resources if the session ends or times out.

Similar to the `ViewState` state bag, session data is stored as a collection of attribute/value pair dictionary entries.

You set and retrieve the dictionary objects using the `Session` keyword, as shown in the next example, which presents a set of radio buttons. Selecting one of the radio buttons sets three session dictionary objects—two strings and a string array. These session dictionary objects are then used to populate a `Label` control and a `DropDownList` control.

Create a new web site called SessionState and switch to the Design view for *default.aspx*. Drag a `RadioButtonList` control onto the page. Set its `AutoPostBack` property to `True`, so the effects will occur as soon as you make a selection.

Use the ListItem Collection Editor to create three items, with their `Text` and `Value` properties set in the following table.

Text	Value
.NET	N
Database	D
Hardware	H

You might also want to set the `RepeatDirection` property to `Horizontal`, and set the `CellSpacing` property to 20, perhaps, to spread things out a bit.

If this were part of a real application, it might make sense to make this an AJAX-enabled web site and wrap all this in an `UpdatePanel` to get much snappier performance. For this example, that is neither necessary nor particularly noticeable.

Drag a `Label` control onto the page and set its ID to `lblMessage`. Clear its `Text` property. Drag a `DropDownList` control onto the page. Set its `ID` to `ddl` and its `Visible` property to `False` so that it will initially be invisible. The Design view will look something like Figure 7-11.

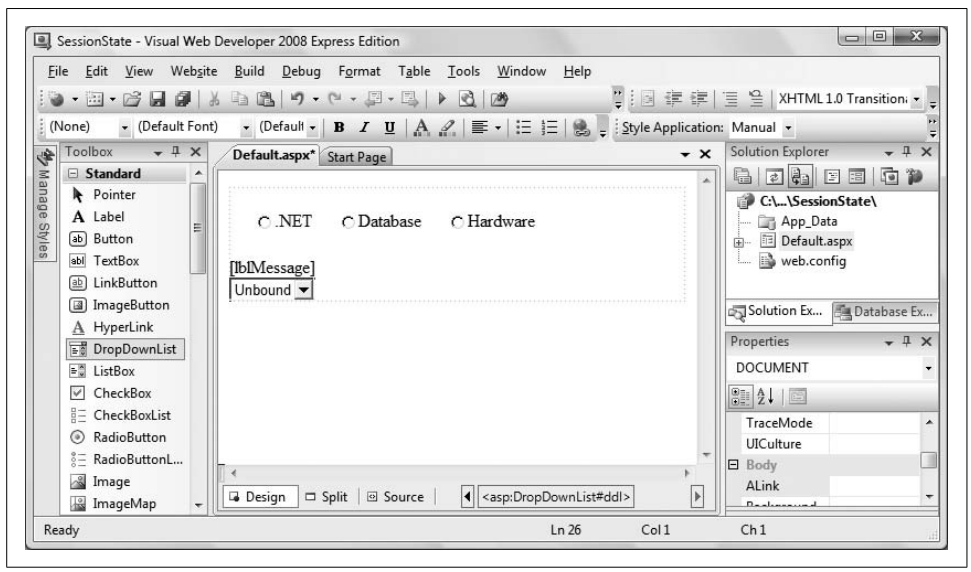


Figure 7-11. *This is what the SessionState Default page should look like in Design view once you've created it. However, populating the drop-down list is different than you might expect.*

Double-click the `RadioButtonList` to open the code-behind in an event handler for the default event for that control, `SelectedIndexChanged`. Enter the code highlighted in Example 7-3. Notice that in addition to the code inside the event handler itself, there is also a helper method called `DisplayStuff`.

Example 7-3. *The Default.aspx.vb for the SessionState web site contains a helper method that can retrieve content directly from the session state dictionary*

```
Partial Class _Default
    Inherits System.Web.UI.Page

Protected Sub RadioButtonList1_SelectedIndexChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles RadioButtonList1.SelectedIndexChanged
    Dim Books(3) As String
    Session("cattext") = RadioButtonList1.SelectedItem.Text
    Session("catcode") = RadioButtonList1.SelectedItem.Value
    Select Case RadioButtonList1.SelectedItem.Value
        Case "N"
            Books(0) = "Programming Visual Basic 2008"
            Books(1) = "Programming ASP.NET"
            Books(2) = "Programming C#"
        Case "D"
            Books(0) = "Oracle & Open Source"
            Books(1) = "SQL in a Nutshell"
            Books(2) = "Transact SQL Programming"
        Case "H"
            Books(0) = "PC Hardware in a Nutshell"
            Books(1) = "Dictionary of PC Hardware and Data Communications Terms"
            Books(2) = "Linux Device Drivers"
    End Select
    Session("books") = Books
    DisplayStuff( )
End Sub

Private Sub DisplayStuff( )
    If RadioButtonList1.SelectedIndex = -1 Then
        lblMessage.Text = "You must select a book category"
    Else
        Dim str As String = String.Empty
        str += "You have selected the category "
        str += CType(Session("cattext"), String)
        str += " with code '"
        str += CType(Session("catcode"), String)
        str += "'."
        lblMessage.Text = str
        ddl.Visible = True
        Dim CatBooks( ) As String = CType(Session("books"), String( ))
         ' populate the DropDownList
        ddl.Items.Clear( )
        For i As Integer = 0 To CatBooks.Length - 1 Step 1
            ddl.Items.Add(New ListItem(CatBooks(i)))
        Next
    End If
End Sub

End Class
```

Run the application and select one of the radio buttons. Then, open the drop-down list to see that the items have been populated, as shown in Figure 7-12. Now, select

one of the other radio buttons. Notice the page posts back immediately, and the content of the drop-down list changes.

VB CHEAT SHEET

Select Case Statement

You saw the `If` statement back in Chapter 4. The `Select Case` statement is a way to string together multiple `If` statements in a clearer manner. The `Select Case` statement opens with the keywords `Select Case`, followed by the variable being evaluated:

```
Select Case myVariable
```

Next is a series of `Case` statements, each with a different condition that will be compared against the value of `myVariable`. Each case is followed by some code that runs if the value of the variable matches the condition. Assuming `myVariable` is a string that represents a size, the `Case` statements might go like this:

```
Case "Small"
  lblMyLabel.Text = "Just a little."
Case "Medium"
  lblMyLabel.Text = "Standard size."
Case "Large"
  lblMyLabel.Text = "Super-size me."
```

Once a case is matched, only the code for that case is executed; the rest of the code is ignored. In Example 7-3 (shown previously), the variable being evaluated is `RadioButtonList1.SelectedItem.Value`, the value that the user chose.

The first thing that happens in this code is the `Text` the user selected is added to session state and associated with the key `"cattext"` in the dictionary. Similarly, the `Value` that goes with that text is stored in `Session` associated with the key `"catcode"`.

```
Session("cattext") = RadioButtonList1.SelectedItem.TextSession("catcode") =
RadioButtonList1.SelectedItem.Value
```

The `Select Case` statement is used to populate the drop-down list, depending on the user's selection. In each case, a three-item array called `Books` is created, but the text for each item varies depending on the `Case` statement. After `Books` is populated, it, too, is saved to Session state:

```
Session("books") = Books
```

Then, the `DisplayStuff( )` helper method is called. Because `cattext`, `catvalue`, and `books` have all been saved in session state, you don't need to pass their values to the helper method. `DisplayStuff( )` can retrieve them directly from the `Session` dictionary, for example, when it concatenates `cattext` to the string:

```
str += CType(Session("cattext"), String)
```

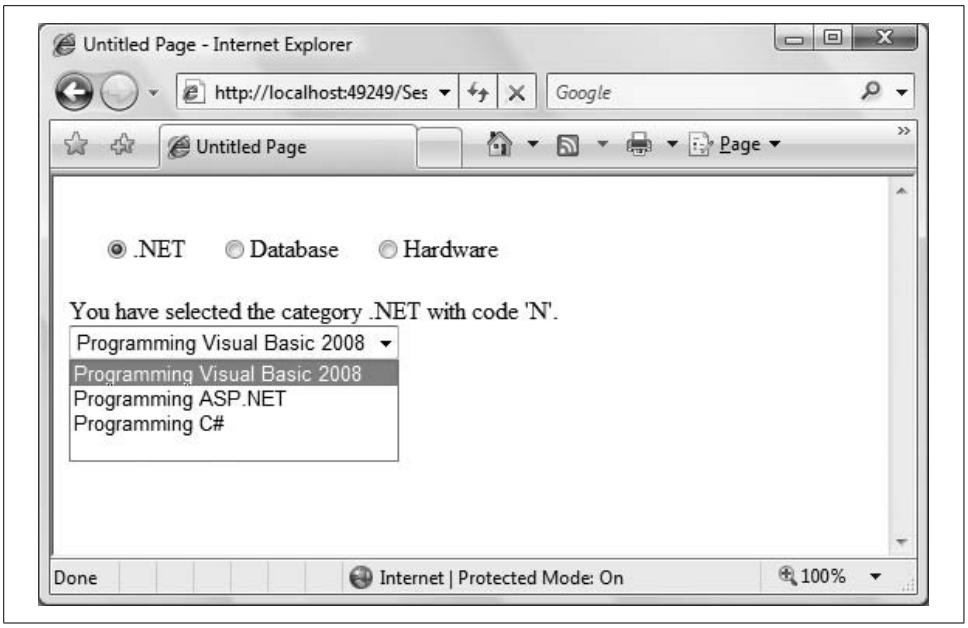


Figure 7-12. *When you select one of the radio buttons in the SessionState web site, the content of the drop-down list changes immediately.*

Remember, you need to use the `CType` method to convert the value to a string before you can use it.

Similarly, the helper method retrieves the `books` object, uses `CType` to convert it to an array of strings, and stores it in the new array `CatBooks()`:

```
Dim CatBooks() As String = CType(Session("books"), String())
```

Next, the method uses `CatBooks()` in a loop to populate the drop-down list.

Session state is enabled by default and works right out of the box. To increase performance, you can disable session state on a per-page, per-web site, or per-machine basis.

To disable session state for a page, include the following highlighted attribute in the `Page` directive at the top of the markup file:

```
<%@ Page Language="VB" AutoEventWireup="false"
    CodeFile="Default.aspx.vb"
    Inherits="_Default"
    EnableSessionState="False" %>
```

Valid values for `EnableSessionState` are `True`, `False`, and `ReadOnly`. `ReadOnly` provides better performance than `True` as long as you do not need to edit the values stored in session. How do the values get into session state if it is set to `ReadOnly`? From another page whose value is `True`.

To configure session state on a per-web site or per-machine basis, you must edit the configuration file, *web.config*, in the web site virtual directory.

For simple single-server, single-processor web sites with relatively low traffic (measured in hits per minute rather than hundreds or thousands of hits per minute), the default configuration is probably good enough. In more complex or demanding scenarios, you can configure session state to accommodate a wide range of requirements. This would include the length of the timeout, whether browser cookies are used, where the session information is stored (in memory on the local machine, in memory on a state server, or in a database somewhere), and so on.

Application State

You have seen how view state, when accessed with the `ViewState` keyword, is scoped to the page. You have also seen that session state, when accessed with the `session` keyword, is scoped to the Session. Finally, there is *application state*, which when accessed with the `Application` keyword, is scoped across the entire application; that is, it applies to all sessions.

The syntax for getting and setting values in `Application` state are exactly analogous to `ViewState` and `Session`, so we will not include an example here. There are several things to consider, however.

First, because multiple sessions can access the same `Application` dictionary object, it is possible for two people to change the value simultaneously. This is not an issue for read-only access. However, if the application data is editable by users, it is possible for two (or more) users to overwrite each other's values, resulting in faulty data at best. To prevent this, you can lock the Application object, but that can cause the application to grind to a halt.

Second, unlike view state and specially configured session state, application state always consumes server memory, so do not store too much "stuff" in application state.

Third, data stored in application state will not survive a server restart or crash. If something needs to persist across application halts, you need to store it elsewhere, such as in *web.config* as an `AppSetting` (but only if it is read-only), or in a database.

And finally, application state is specific to a single processor on a single machine. If your environment includes either multiprocessor servers (web garden) or multiple web servers (web farm), do not use application state. If you need this type of functionality, you will need to create it from scratch, perhaps storing the requisite values in a database.

Summary

- When you issue a request from your browser, that request is sent across the Internet to a remote server, which processes the request, possibly accesses a database, and then returns the HTML to the browser, where the page is rendered.
- A session is the period where a single user interacts with the web application, no matter how many different pages are visited in that application.
- The values of all the controls on the page are referred to as state. The Web was originally intended not to preserve state, but that has evolved over time.
- Each step of the ASP.NET life cycle has events that allow you to change or add to the default behavior for that step.
- The control tree is a hierarchical representation of all the controls on a single page.
- The `Trace` attribute lets you see a great deal of information about your page, including the control tree.
- The page life cycle differs slightly depending on whether the page is loaded as a result of a postback.
- The Start phase determines whether the page was requested by another page, or is a postback.
- The Initialization phase applies personalization and themes to the page, and also reads or initializes control properties.
- The Load phase sets the control properties.
- The Validation phase is where validation methods are checked on all eligible controls on the page.
- The Rendering phase is where each control is called to render itself in the browser.
- The Unload phase is last, and allows you to clean up any resources you need to.
- ASP.NET can automatically maintain state for server controls, avoiding the need to write any custom code.
- View state is the state of the page and any controls on it.
- You can disable the view state by adding `EnableViewState="false"` to the Page directive. Simple controls such as text boxes, radio buttons, and checkboxes always preserve view state, no matter what you set in the `Page` directive. More complex controls, however, can have their view state disabled.
- Although the view state is maintained automatically, you can use the state bag of view state to store your own values from post to post, using the `ViewState` keyword.

- The state bag uses a dictionary structure, with keys and values that you define as needed.
- Session state is not automatically maintained from page to page, but you can specify that objects be preserved in session state.
- Session state also uses a dictionary that you define as needed. You use the `Session` keyword to add items to this dictionary.
- You can disable session state for the page by placing the attribute `EnableSessionState="False"` in the `Page` directive, or for the entire site by editing *web.config*.
- Application state is similar to view state and session state. You can access the dictionary with the `Application` keyword.

By now, you've developed a lot of skills and created many web pages that look and act professional, both up front and under the hood. When you take your shiny web site with its fancy controls out for a spin in the real world, though, you're going to come up against a tough reality: stuff breaks…a lot. As you may have already discovered from the exercises in this book, a typo in the wrong place or a misconfigured property can lead to a site that doesn't work and a lot of time staring at the code trying to figure out where you went wrong. Even when all of your code works perfectly, you can still run into problems with outside data sources, user errors, and other things beyond your control. Errors and bugs are part of programming, and nobody expects you to write perfect code the first time. What you want to learn, though, is how to find and fix bugs in the shortest possible time. The IDE has a host of tools to help you do just that, and that's what we'll discuss next.

Quiz

1. What is a session?
2. What is the state of a page?
3. What setting can you use to see information about the different stages of the page life cycle?
4. In which life cycle phase does the page determine if it was called as the result of a postback?
5. What event is usually used to take actions during the Load phase?
6. What are the four kinds of state that ASP.NET manages? Which one can you not affect in any way?
7. What does the `EnableViewState="false"` setting do?
8. Where would you store the value of a counter that is incremented each time the page is loaded?
9. Suppose you wanted to ask the user to enter her name on a page, and you wanted to retain that value the entire time the user is at your site. What's the best mechanism to use?
10. What's the proper syntax for storing and retrieving that user name?

Exercises

Exercise 7-1. You'll start out with a simple exercise that uses your knowledge of the page life cycle. Create a simple page with a label, `lblPostBack`, and a button, `btnPostBack`, with a `Text` property of "Post Back". Write the appropriate code to cause the label to display the message "You're seeing this page for the first time!" when the page initially loads. Whenever the page is loaded as a result of a postback, the label should display the message "Welcome back to the page."

Exercise 7-2. Create a page with a label, `lblMessage`, and a button, `btnPostBack`. The first time you access the page, the label should output the message "Page first accessed at," followed by the date and time. Each time the button is clicked, use the state bag to add a new line to the label with the message "Page posted back at," followed by the date and time. (Hint: If you don't recall how to access the current date and time, see Chapter 3.) Your page should look something like Figure 7-13.

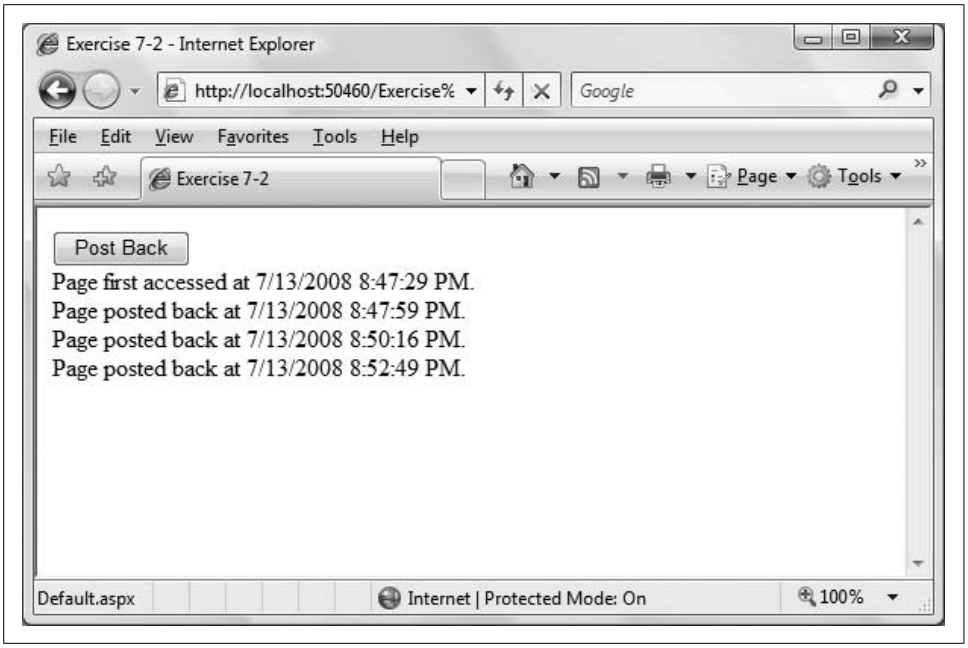


Figure 7-13. *Your goal for Exercise 7-2, after clicking the Post Back button a few times.*

Exercise 7-3. Copy Exercise 7-2 to a new web site, and make the *Default.aspx* page the home page. Add two more pages to the project; call them *SecondPage.aspx* and *ThirdPage.aspx*. Add an `<h1>` to *Default.aspx* to identify it, and then add two navigation buttons to navigate to each of the other two pages.

SecondPage.aspx and *ThirdPage.aspx* should each contain an `<h1>` to identify them, a button for posting back, two buttons for navigating to each of the other two pages, and a label that displays the string created on the home page. Both of these pages should add a message to the string indicating when they were accessed for the first time and when they are posted back. After you've navigated around the site for a bit, it should look like Figure 7-14.

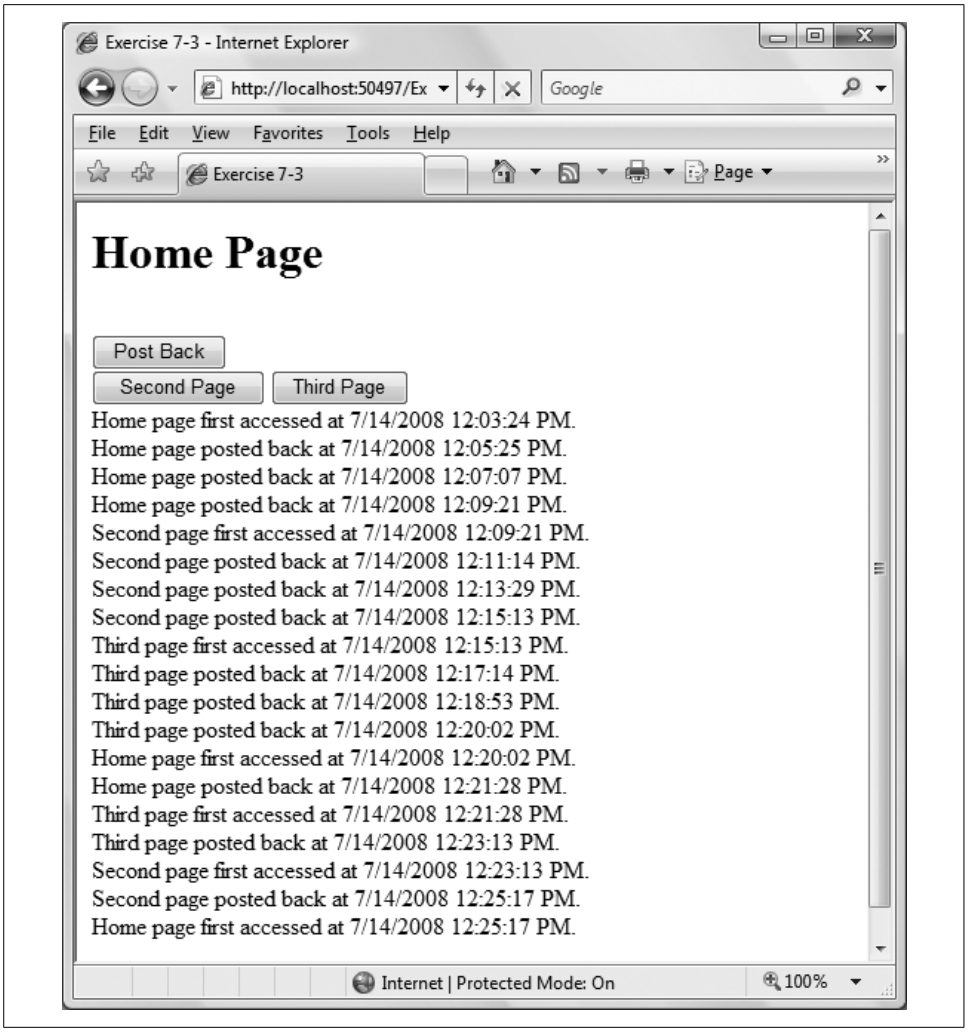

Exercise 7-3 - Internet Explorer
http://localhost:50497/Ex
Google
File Edit View Favorites Tools Help
Exercise 7-3
Page
Home Page
Post Back
Second Page
Third Page
Home page first accessed at 7/14/2008 12:03:24 PM.
Home page posted back at 7/14/2008 12:05:25 PM.
Home page posted back at 7/14/2008 12:07:07 PM.
Home page posted back at 7/14/2008 12:09:21 PM.
Second page first accessed at 7/14/2008 12:09:21 PM.
Second page posted back at 7/14/2008 12:11:14 PM.
Second page posted back at 7/14/2008 12:13:29 PM.
Second page posted back at 7/14/2008 12:15:13 PM.
Third page first accessed at 7/14/2008 12:15:13 PM.
Third page posted back at 7/14/2008 12:17:14 PM.
Third page posted back at 7/14/2008 12:18:53 PM.
Third page posted back at 7/14/2008 12:20:02 PM.
Home page first accessed at 7/14/2008 12:20:02 PM.
Home page posted back at 7/14/2008 12:21:28 PM.
Third page first accessed at 7/14/2008 12:21:28 PM.
Third page posted back at 7/14/2008 12:23:13 PM.
Second page first accessed at 7/14/2008 12:23:13 PM.
Second page posted back at 7/14/2008 12:25:17 PM.
Home page first accessed at 7/14/2008 12:25:17 PM.
Internet | Protected Mode: On
100%

Figure 7-14. *Your goal for Exercise 7-3, after clicking around a bit.*

CHAPTER 8

Errors, Exceptions, and Bugs, Oh My!

Every computer programmer has run into bugs. It comes with the territory. Many bugs are found during the coding process. Others pop up only when an end user performs a specific and unusual sequence of steps or the program receives unexpected data. You should always try to find bugs early in the development process and avoid having end users find your bugs for you. Countless studies have shown that the earlier you find a bug, the easier and less expensive it is to fix.

If your program does run into a problem, you will want to recover quickly and invisibly, or, at worst, fail gracefully. ASP.NET provides tools and features to help reach these goals:

Tracing
: You can trace program execution at either the page or application level. ASP.NET provides an extensible trace log with program life-cycle information.

Symbolic debugging
: You can step through your program line by line, stop it as it is running, examine and modify variables and expressions, and step into and out of classes, even those written in other languages.

Error handling
: You can handle standard or custom errors at the application or page level and display special error pages. You can also show different error pages for different errors.

There's an additional technique for handling errors in your code: the `try/catch` block. A `try` block surrounds code that has a reasonable chance of causing a problem, such as a file not being found when you try to open it. The corresponding `catch` block contains code to handle that error appropriately. You may see `try/catch` blocks in various places in this book, and you can find out more about them in the "Catching Errors" Cheat Sheet in Chapter 6.

To get started exploring the ASP.NET debugging tools, you should first create a simple web site to which you will add tracing code. You will then introduce bugs into the program and use the debugger to find and fix the bugs.

Creating the Sample Application

To start, create a new web site and name it DebuggingApp. This will consist of a single web page containing a header label, a `DropDownList` with a label below it to display the selected item, and a hyperlink.

In Design view, drag a `Label` control to the top of the page and set its `Text` property to the following:

```
Tracing, Debugging & Error Handling Demo
```

Change its `Font-Name` property to `Arial Black`, its `Font-Size` property to `Large`, and its `Font-Bold` property to `True`.

Drag a `DropDownList` control onto the form. Set its ID property to `ddlBooks`. Change its `AutoPostBack` property to `True`.

Add a label below the `DropDownList` with an ID of `lblDdl`. Set the `Text` property so it is empty.

Finally, add a `HyperLink` control below `lblDdl`. Set its ID property to `hlTest`. Change the `Text` property to "Link To" and change the `NavigateUrl` property to *TestLink.aspx*. No page with this name exists. This is an intentional error to demonstrate error handling later in the chapter.

The Design view will look something like that shown in Figure 8-1.

The `DropDownList` will be dynamically populated every time the page loads in the `Page_Load` method. To create this method, double-click *Default.aspx.vb* in the Solution Explorer to open the code-behind file. Click the Class Name drop-down at the upper left of the editing window and select (`Page Events`).

Click the Method Name drop-down on the upper right and select the `Load` event to create the code skeleton for the `Page_Load` method. Enter the highlighted code from Example 8-1 into the `Page_Load` method.

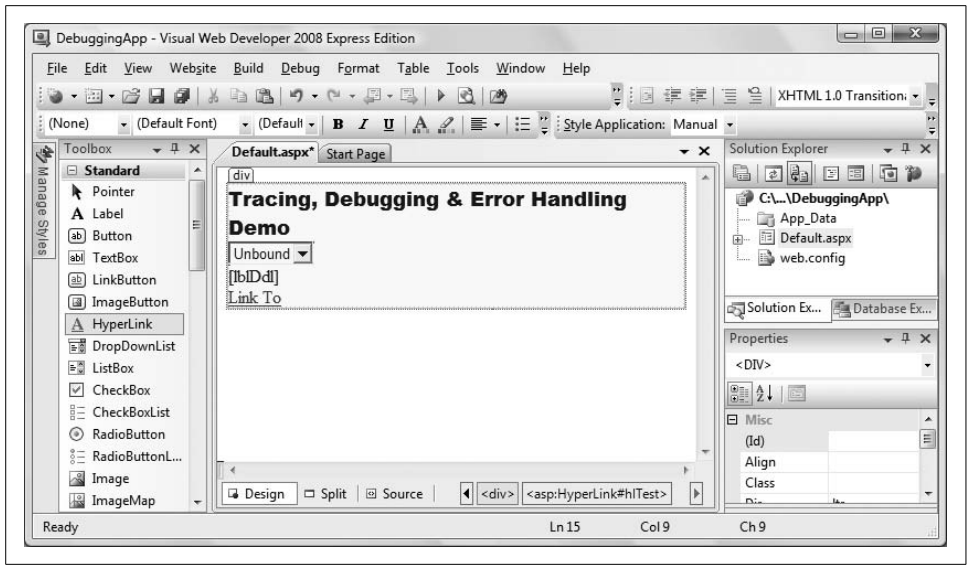


Figure 8-1. *Here's what the sample application, DebuggingApp, looks like in Design view.*

Example 8-1. *The Page_Load method for the Default page in DebuggingApp defines an array of books*

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
   If Not IsPostBack Then
       ' Build 2 dimensional array for the lists
       ' First dimension contains bookname
       ' Second dimension contains ISBN number
       Dim books(,) As String = { _
            {"Programming Silverlight", "0000000001"}, _
            {"Programming .NET 3", "0000000002"}, _
            {"Programming ASP.NET, 4th Edition", "0000000003"}, _
            {"Programming Visual Basic 9", "0000000004"}, _
            {"Programming C#, 5th Edition", "0000000005"}, _
            {"Learning ASP.NET ", "0596513976"} _
       }
         ' Now populate the list
       For i As Integer = 0 To books.GetLength(0) - 1
           '  add both Text and Value
           ddlBooks.Items.Add(New ListItem(books(i, 0), books(i, 1)))
        Next
    End If
End Sub
```

There is no need to create the array of book names and populate the DropDownList on every page load because view state will handle that on postback. Therefore, you test the IsPostBack property and only create the array the first time the page is loaded.

Now, you need to add the event-handling code for the drop-down list. In Design view, double-click the control to open the code-behind file, *Default.aspx.vb*. The cursor will be in the event handler method ddlBooks_SelectedIndexChanged. Type in the highlighted code from Example 8-2. You could also accomplish this check with the validation controls you learned about in Chapter 5.

Example 8-2. *The SelectedIndexChanged event handler for ddlBooks simply relays the user's choice to the label*

```
Protected Sub ddlBooks_SelectedIndexChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles ddlBooks.SelectedIndexChanged
    ' check to verify that something has been selected
    If ddlBooks.SelectedIndex <> -1 Then
        lblDdl.Text = ddlBooks.SelectedItem.Text + " ---> ISBN: " _
            + ddlBooks.SelectedItem.Value
    End If
End Sub
```

Run the app and select a book title from the DropDownList. You will see something similar to Figure 8-2.

Figure 8-2. *Here's DebuggingApp in action. After you select a book title from the DropDownList, its title and ISBN are output in the Label.*

You will use this application through the rest of this chapter to demonstrate various techniques for analyzing and debugging code in ASP.NET and for handling errors in your application.

VB CHEAT SHEET

For Loops

You saw the `For Each` loop back in Chapter 6; the `For` loop is simply a more general version. The first line of a `For` loop has three parts: the condition, the beginning, and the end. Usually, you create a loop control variable, an integer, often named `i`, to be the condition. For example, if you wanted a loop that would run 10 times, you would start it like this:

```
For i As Integer = 1 To 10
```

Inside the loop, you place whatever code you want to run on each iteration. Then, you end the loop with a `Next` statement. That causes execution to return to the beginning of the loop until the condition reaches the upper limit.

You can also use your control variable within the loop, as it increments. For example, this loop would add the numbers 1 through 10 to `myString`:

```
For i As Integer = 1 To 10
   myString = myString + i + " "
Next
```

This works because `i` will be incremented each time through the loop.

Usually, you don't know exactly how many times you want your loop to run, so you set the upper limit at runtime. In Example 8-1 (shown earlier in this chapter), you want to loop through each item in the `books` array. Array indexes start at 0, as we've mentioned, and you use the `GetLength( )` method on the array to find out how many items are contained in it. However, the length of the array is measured from 1, not 0, so you have to use `GetLength( ) -1` to find the correct upper boundary for the loop:

```
For i As Integer = 0 To books.GetLength(0) - 1
```

Notice that in the books example, `i` is also used within the loop to extract the item with the index of `i` from `books`, and add it to the `ddlBooks` collection of `ListItems`.

Tracing

Tracing is the technique of reporting the value or state of things in your program as it runs. It is an easy way to determine what is going on in your program. Back in the days of classic ASP, the only way to trace what was happening in your code was to insert `Response.Write` statements in strategic places. This allowed you to see that you had reached a known point in the code and, perhaps, to display the value of some variables. The big problem with this hand-tracing technique, aside from the amount of work involved, was that you had to laboriously remove or comment out all those

statements before the program went into production. ASP.NET provides better and easier ways of gathering the trace information.

Page-Level Tracing

As you saw in Chapter 7, to add page-level tracing, modify the `Page` directive at the top of the *.aspx* page (in Source view) by adding a `Trace` attribute and setting its value to `true`, as follows (remember to close the browser first):

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
       Inherits="_Default" Trace="true" %>
```

Run the page. When you view this page, tables will be appended to the bottom of the page that contain a wealth of information about your web application. Select a book from the drop-down list and you will see something like Figure 8-3.

The top section, labeled Request Details, shows information, including the SessionID, the Time of Request, Request Type, and Status Code (see Table 8-1). Every time the page is posted to the server, this information is updated. If you change the selection (remember that `AutoPostBack` is set to true), you will see that the Time of Request is updated, but the `SessionID` remains constant.

Table 8-1. *Commonly used HTTP status codes*

Category	Number	Description
Informational (100–199)	100	Continue
Successful (200–299)	200	OK
Client Errors (400–499)	400	Bad request
	401	Unauthorized
	403	Forbidden
	404	Not found
	408	Request timeout
Server Error (500–599)	500	Internal Server Error

Every web user has seen at least some of these status codes in their normal browsing, including the ubiquitous "404 -Not Found". For a complete list, go to *http://en.wikipedia.org/wiki/List_of_HTTP_status_codes*.

The next section of the trace, labeled "Trace Information," is the *trace log* (this section was shown briefly in Chapter 7 in the discussion of life cycle), which provides life-cycle information. This includes elapsed times, in seconds, since the page was initialized (the From First(s) column) and since the previous event in the life cycle (the From Last(s) column). You can add custom trace information to the trace log, as explained shortly.

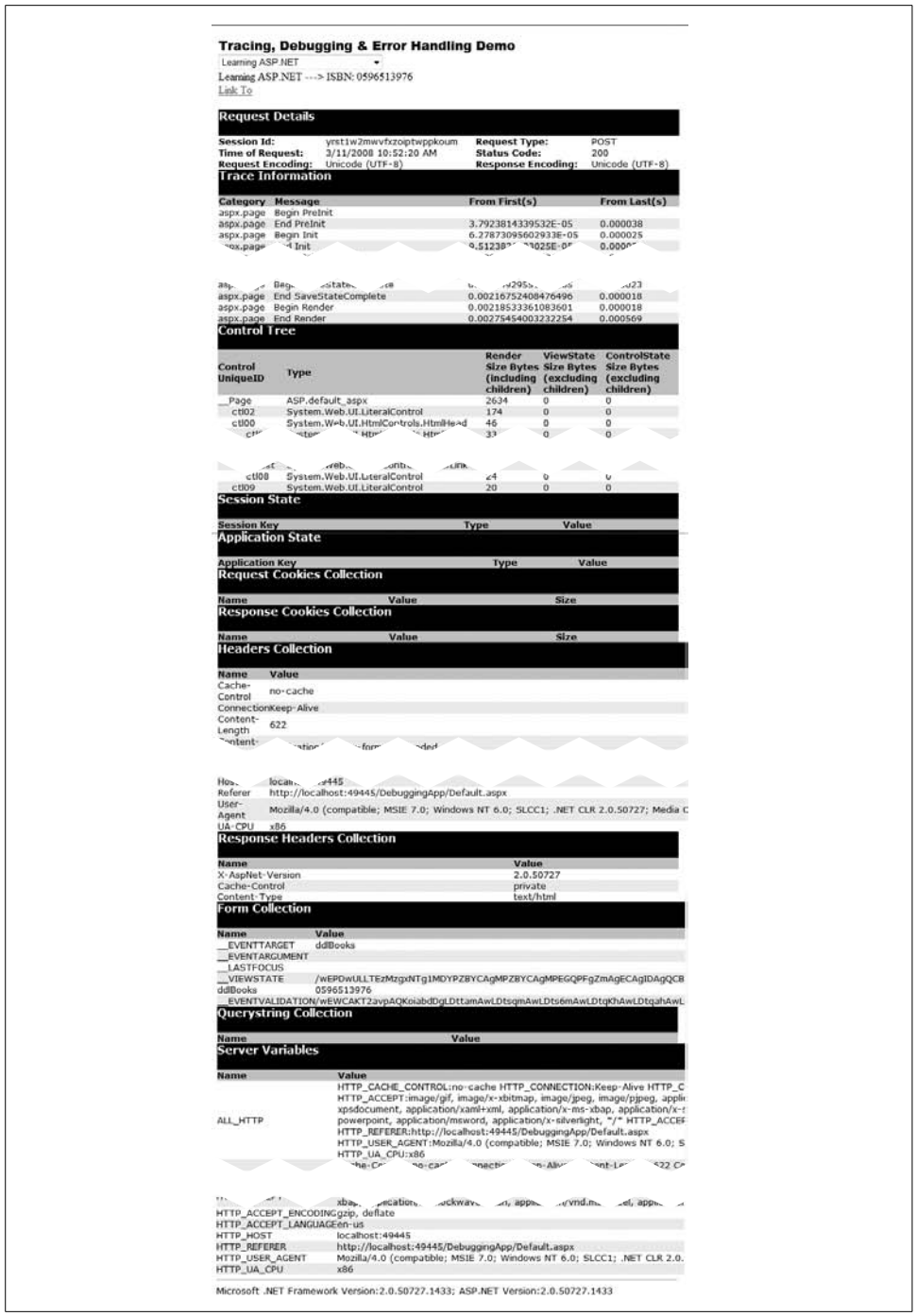


Figure 8-3. *When you enable tracing in your file, you get a lot of data as a result.*

The next section in the trace, under the heading Control Tree, lists all the controls on the page in a hierarchical manner, including the name of the control, its type, and its size in bytes, on the page, in the `ViewState` state bag, and in control state.

This is followed by Session and Application State summaries, and itemizations of the Cookies and Headers collections. Finally, there is a list of all the server variables.

Inserting into the Trace Log

You can add custom information to the trace output with methods of the `Trace` object. This object exposes two methods for putting your own statements into the trace log: `Write` and `Warn`. The only difference between the two methods is that `Warn` writes to the log in red. The `Warn` and `Write` methods can take either a single argument, two arguments, or two strings and an exception object (generated by the .NET Framework when you use a `try/catch` block), as the following cases illustrate:

`Trace.Warn("Warning Message")`
: Inserts a record into the trace log with the message passed in as a string.

`Trace.Warn("Category","Warning Message")`
: Inserts a record into the trace log with the category and message you pass in.

`Trace.Warn("Category","Warning Message", excp)`
: Inserts a record into the trace log with a category, warning message, and exception.

To see this in action, add the highlighted code from Example 8-3 to the code-behind file in your sample web site, DebuggingApp.

Example 8-3. *Writing to the Trace object is very simple; just use the Trace.Write() method*

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    Trace.Write("In Page_Load")
    If Not IsPostBack Then
       Trace.Write("Page_Load", "Not PostBack.")
       ' Build 2 dimensional array for the lists
       ' First dimension contains bookname
       ' Second dimension contains ISBN number
       Dim  books(,) As String = { _
            {"Programming Silverlight", "0000000001"}, _
            {"Programming .NET 3", "0000000002"}, _
            {"Programming ASP.NET, 4th Edition", "0000000003"}, _
            {"Programming Visual Basic 9", "0000000004"}, _
            {"Programming C#, 5th Edition", "0000000005"}, _
            {"Learning ASP.NET ", "0596513976"} _
     }
       '    Now populate the list
       For i As Integer = 0 To books.GetLength(0) - 1
           '  add both Text and Value
           ddlBooks.Items.Add(New ListItem(books(i, 0), books(i, 1)))
```

Example 8-3. *Writing to the Trace object is very simple; just use the Trace.Write() method (continued)*

```
        Next
    End If
End Sub

Protected Sub ddlBooks_SelectedIndexChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles ddlBooks.SelectedIndexChanged
    '  force an exception
    Try
       Dim a As Integer = 0
       Dim b As Integer = 5 / a
    Catch ex As Exception
       Trace.Warn("User Action", "Calling b=5/a", ex)
    End Try

    ' check to verify that something has been selected
    If ddlBooks.SelectedIndex <> -1 Then
       lblDdl.Text = ddlBooks.SelectedItem.Text + " ---> ISBN: " _
           + ddlBooks.SelectedItem.Value
    End If
End Sub
```

The first message is added in the `Page_Load` method to signal that you've entered that method:

```
Trace.Write("In Page_Load")
```

The second message is added if the page is not a postback:

```
If Not IsPostBack Then
    Trace.Write("Page_Load", "Not PostBack.")
```

This second message is categorized as `Page_Load`; using a category can help you organize the trace output. Run the app to see the effect of these two `Write` statements, as shown in Figure 8-4.

The third message is added to demonstrate the process of inserting an exception into the error log. The `ddlBooks_SelectedIndexChanged` event handler now contains code to force an exception by dividing by zero. The code catches that exception and logs the exception with a `Trace` statement, as shown by the following code fragment:

```
Try
    Dim a As Integer = 0
    Dim b As Integer = 5 / a
Catch ex As Exception
    Trace.Warn("User Action", "Calling b=5/a", ex)
End Try
```

The output from this `Trace` statement is shown in Figure 8-5.

Because this `Trace` statement was written by calling the `Warn` method rather than the `Write` method, the trace output appears in red onscreen (though not in your printed

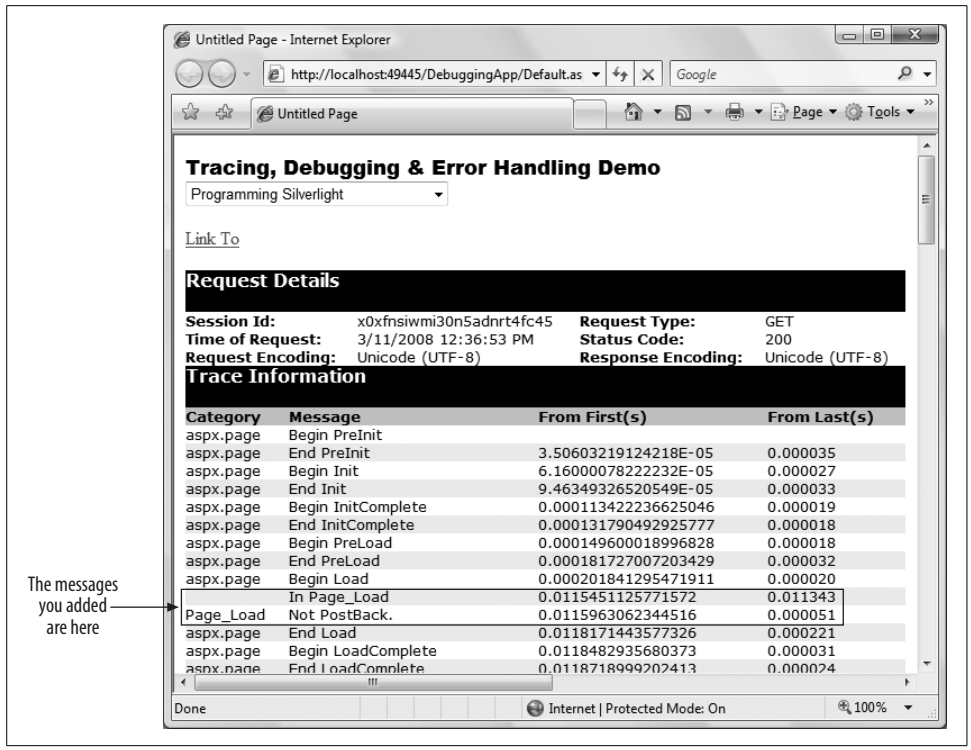


Figure 8-4. *You can insert Trace.Write statements in your code to send specific messages to the trace output.*

copy of this book). Notice that the string you passed in, `Calling b=5/a`, is displayed, followed by an error message extracted automatically from the exception object.

Implementing `Trace` statements is easy, and when it's time to put your page into production, all these statements can remain in place. The only modification you need to make is to change the `Trace` attribute in the `Page` directive from `true` to `false`.

Debugging

Tracing provides you with a snapshot of the steps your code has taken after the code has run. At times, however, you'd like to monitor your code while it is running. What you want is more of a CAT scan than an autopsy. The code equivalent of a CAT scan is a *symbolic debugger*.

When you run your code in the debugger, you can watch your code work, step by step. As you walk through the code, you can see the variables change values, and you can watch as objects are created and destroyed.

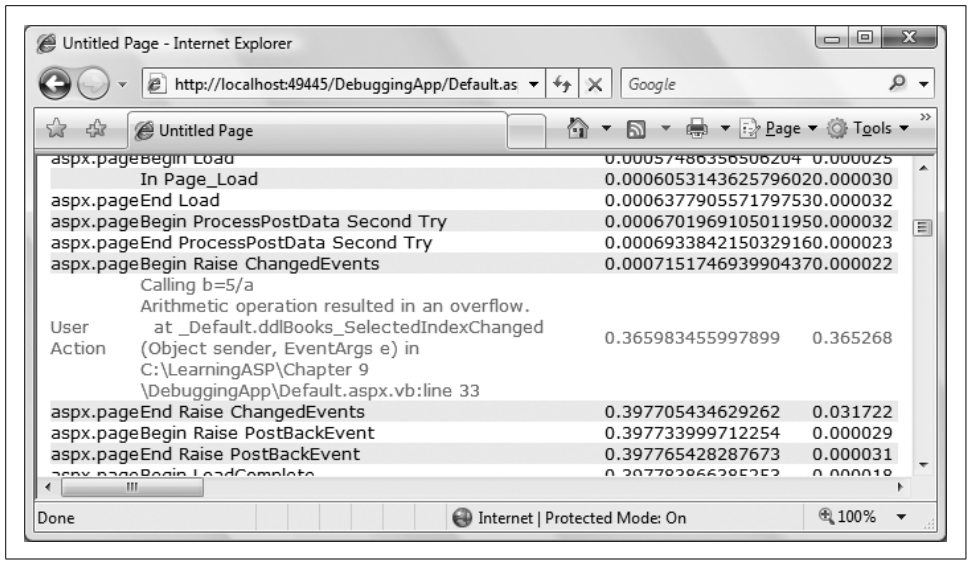

Untitled Page - Internet Explorer
http://localhost:49445/DebuggingApp/Default.as
Google
Untitled Page
Page
Tools
In Page_Load 0.000605314362579602 0.000030
aspx.page End Load 0.000637790557179753 0.000032
aspx.page Begin ProcessPostData Second Try 0.000670196910501195 0.000032
aspx.page End ProcessPostData Second Try 0.000693384215032916 0.000023
aspx.page Begin Raise ChangedEvents 0.000715174693990437 0.000022
Calling b=5/a
Arithmetic operation resulted in an overflow.
User Action at _Default.ddlBooks_SelectedIndexChanged (Object sender, EventArgs e) in C:\LearningASP\Chapter 9 \DebuggingApp\Default.aspx.vb:line 33
0.365983455997899 0.365268
aspx.page End Raise ChangedEvents 0.397705434629262 0.031722
aspx.page Begin Raise PostBackEvent 0.397733999712254 0.000029
aspx.page End Raise PostBackEvent 0.397765428287673 0.000031
Done
Internet | Protected Mode: On
100%

Figure 8-5. *When you use Trace.Warn to indicate a caught exception, it looks like this. The Trace.Warn output displays in red.*

This section will provide a brief introduction to the most important parts of the debugger that accompanies the VS and VWD IDE. For complete coverage of how to use the debugger, we urge you to spend time with the documentation and to experiment freely. The debugger is one of the most powerful tools at your disposal for learning ASP.NET.

You can configure an application to enable or disable debugging through the configuration file, *web.config*. As you've already seen many times, the first time you run an application, a dialog box will pop up, offering to automatically make the necessary edits to *web.config* to enable debugging.

The debugging configuration information is contained within the `<compilation>` section, within the `<system.web>` section, which in turn is contained within the `<configuration>` section. So, a typical compilation configuration snippet will look something like Example 8-4.

Example 8-4. *This debug configuration code snippet from web.config doesn't look like much, but it makes a big difference in your debugging efforts*

```
<?xml version="1.0"?>
<configuration>

   <system.web>
.
.
.
     <compilation debug="true" strict="false" explicit="true">
.
```

Example 8-4. *This debug configuration code snippet from web.config doesn't look like much, but it makes a big difference in your debugging efforts (continued)*

```
.
.
    </compilation>
  />
```

Debugging is strictly for when you're developing and testing your application. When you put your web site into production, you should disable debugging. Setting `debug` to `false` improves the runtime performance of the application, and it suppresses certain output that you probably don't want your users to see.

The Debug Toolbar

A *Debug toolbar* is available in the IDE. To make it visible, click the View → Toolbars menu commands, and then click Debug, if it is not checked. Table 8-2 shows the icons that appear on the Debug toolbar. The use of most of these buttons may not be obvious to you right now, but you'll see how they work when we talk about breakpoints shortly.

Table 8-2. *Debug toolbar icons*

Icon	Debug menu equivalent	Keyboard shortcut	Description
	Start/Continue	F5	Starts or continues executing the program.
	Break All	Ctrl-Alt-Break	Stops program execution at the currently executing line.
	Stop Debugging	Shift-F5	Stops debugging.
	Restart	Ctrl-Shift-F5	Stops the run currently being debugged and immediately begins a new run.
			Shows the next statement that will be executed.
	Step Into	F11	If the current line contains a call to a method or function, this icon will single-step the debugger into that method or function.
	Step Over	F10	If the current line contains a call to a method or function, this icon will not step into that method or function, but will go to the next line after the call.

Table 8-2. *Debug toolbar icons (continued)*

Icon	Debug menu equivalent	Keyboard shortcut	Description
	Step Out	Shift-F11	If the current line is in a method or function, that method or function will complete and the debugger will stop on the line after the method or function call.
			Hexadecimal display toggle.
	Output		Debug window selector.

Breakpoints

The crux of the biscuit is the apostrophe.
—Frank Zappa, "Apostrophe (')"

Breakpoints are at the heart of debugging. A breakpoint is an instruction to the debugger to run to a specific line in your code and to stop and wait for you to examine the current state of the application. When the execution is paused, you can do the following:

- Examine and modify values of variables and expressions.
- Single-step through the code.
- Move into and out of methods and functions, even stepping into classes written in other .NET languages compliant with the Common Language Runtime.
- Perform any number of other debugging and analysis tasks.

Setting a breakpoint

You can set a breakpoint in the code-editing window when you're editing a .NET-compliant language, such as VB or C#, by single-clicking the gray vertical bar along the left margin of the window. You can also place the mouse cursor on the line of code and press F9 to toggle a breakpoint on or off. In either case, if a breakpoint is set for a line, a red dot will appear in the left margin and the line of code will be highlighted, as shown in Figure 8-6.

Breakpoints window

You can see all the breakpoints currently set by looking at the *Breakpoints window*.

For some strange reason, the Breakpoints window is available in Visual Studio 2008, but not in Visual Web Developer (VWD) 2008.

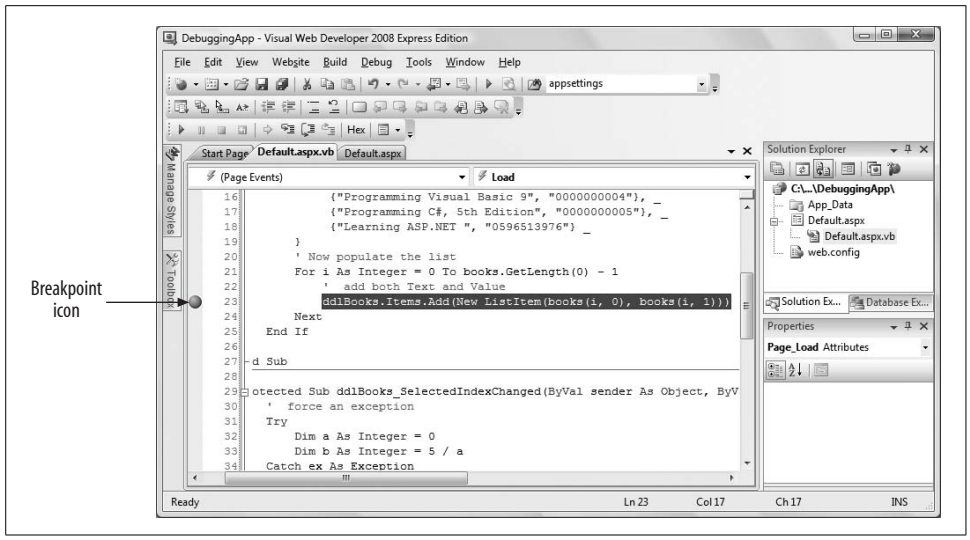


Figure 8-6. *Setting a breakpoint in a code-editing window is simple; just click in the left margin of your code or press F9.*

To display the Breakpoints window, perform any one of the following actions:

- Press Ctrl-Alt-B.
- Select Breakpoints from the Debug → Windows menu command.
- Click the Windows icon of the Debug toolbar and select Breakpoints.

A Breakpoints window is shown in Figure 8-7.

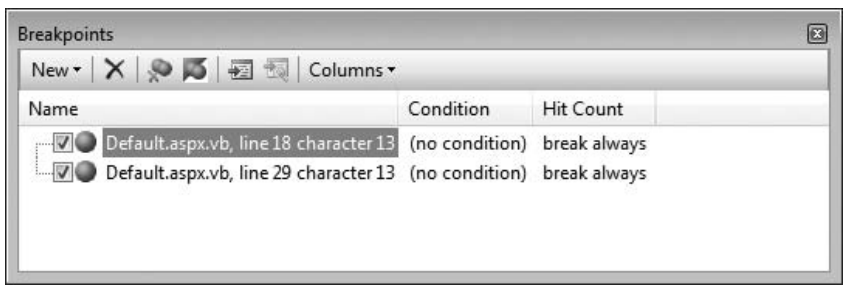


Figure 8-7. *The Breakpoints window shows you all the breakpoints you currently have set. Unfortunately, this window is only available in Visual Studio 2008.*

You can toggle a breakpoint between Enabled and Disabled by clicking the corresponding checkbox in the Breakpoints window.

Breakpoint properties

Sometimes you don't want a breakpoint to stop execution every time the line is reached. VS2008 offers several properties that can be set to modify the behavior of a breakpoint. You can set these properties via the property menu, arrived at in either of two ways:

- Right-click the breakpoint glyph in the left margin.
- Open the Breakpoints window and right-click the desired breakpoint.

Again, this feature is available only in Visual Studio 2008, not VWD. However, right-clicking the breakpoint glyph in the left margin in VWD will offer the choice of deleting or disabling the breakpoint.

In either case, you will see the context menu shown in Figure 8-8.

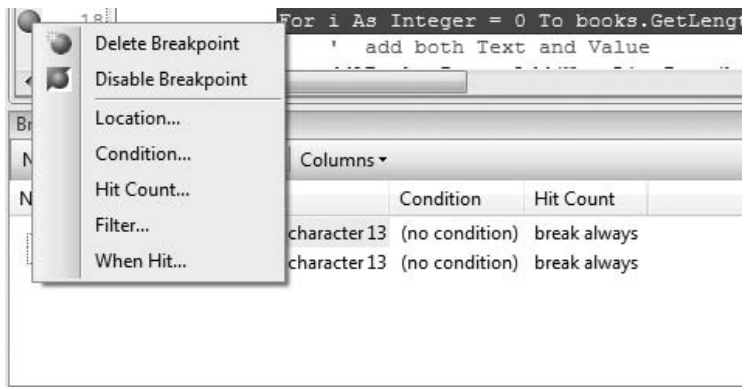


Figure 8-8. *When you right-click on a breakpoint, you'll see this properties menu, where you can delete or disable a breakpoint, and also make a number of refinements to the breakpoint's behavior.*

The first two items in the breakpoint properties menu allow you to delete or disable the selected breakpoint. The Disable menu item will toggle each time you click it, and when the breakpoint is disabled, the icon will appear as an empty circle. The following menu items are available:

Location. The Location menu item brings up the dialog box shown in Figure 8-9, which is fairly self-explanatory. Using this dialog box is equivalent to setting a breakpoint in the code window, with a few additional options.

Condition. The Condition button brings up the dialog box shown in Figure 8-10.

You can enter any valid expression in the edit field. This expression is evaluated when program execution reaches the breakpoint. Depending on which radio button is selected and how the Condition expression evaluates, the program execution will either pause or move on. The two radio buttons are labeled as follows:

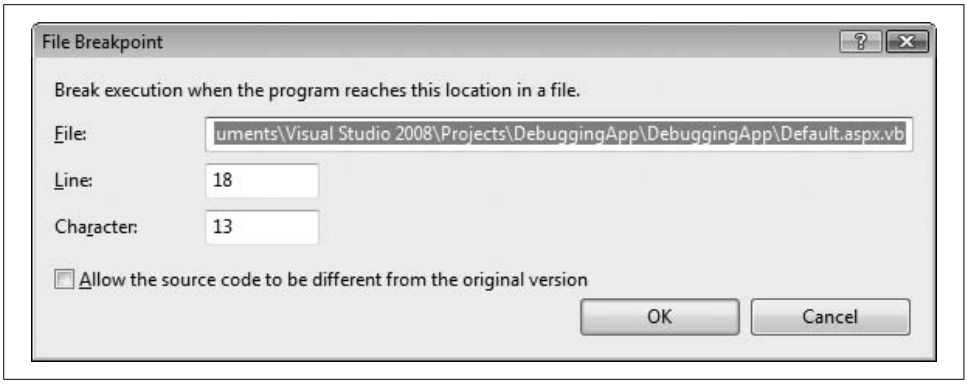


Figure 8-9. *You can use the File Breakpoint dialog box to set a breakpoint at a specific spot in your file.*

Breakpoint Condition
When the breakpoint location is reached, the expression is evaluated and the breakpoint is hit only if the expression is true or has changed.
Condition:
Is true
Has changed
OK
Cancel

Figure 8-10. *You can use the Breakpoint Condition dialog box to specify when you want the breakpoint to stop execution.*

Is true
: If the Condition entered evaluates to a Boolean `true`, the program will pause.

Has changed
: If the Condition entered has changed, then the program will pause. On the first pass through the piece of code being debugged, the breakpoint will never pause execution because there is nothing to compare against. On the second and subsequent passes, the expression will have been initialized and the comparison will take place.

Hit count. Hit count is the number of times that spot in the code has been executed since either the run began or the Reset Hit Count button was pressed. The Hit Count button brings up the dialog box shown in Figure 8-11.

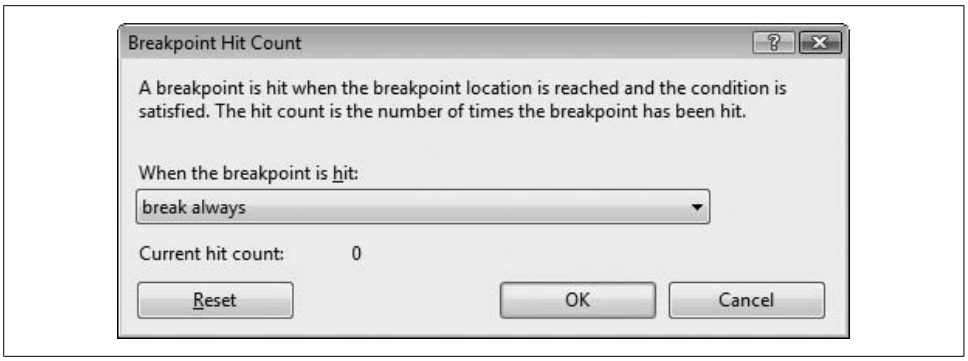

Breakpoint Hit Count
A breakpoint is hit when the breakpoint location is reached and the condition is satisfied. The hit count is the number of times the breakpoint has been hit.
When the breakpoint is hit:
break always
Current hit count: 0
Reset
OK
Cancel

Figure 8-11. *You can use the Breakpoint Hit Count dialog box to cause the breakpoint to only activate at set intervals.*

Clicking the drop-down list presents the following options:

- Break always
- Break always when the hit count is equal to
- Break always when the hit count is a multiple of
- Break always when the hit count is greater than or equal to

If you click any option other than "break always" (the default), the dialog box will add an edit field for you to enter a target hit count.

Suppose this is a breakpoint set in a loop of some sort. You select "break when the hit count is a multiple of" and enter 5 in the edit field. The program will pause execution every fifth time it runs.

When a hit count is set, the red breakpoint icon in the left margin of the window has a plus sign in the middle of it.

When Hit…. The When Hit menu item brings up the dialog box shown in Figure 8-12. By default, the "Print a message" checkbox is unchecked. When this box is checked, the red circular breakpoint icon in the left margin of the window changes to a diamond shape.

You can also elect to run one of a large selection of predefined macros, such as FindCase, SaveView, and SaveBackup.

By default, the Continue execution checkbox is checked.

Breakpoint icons

Each breakpoint symbol, or *glyph*, conveys a different type of breakpoint. These glyphs appear in Table 8-3.

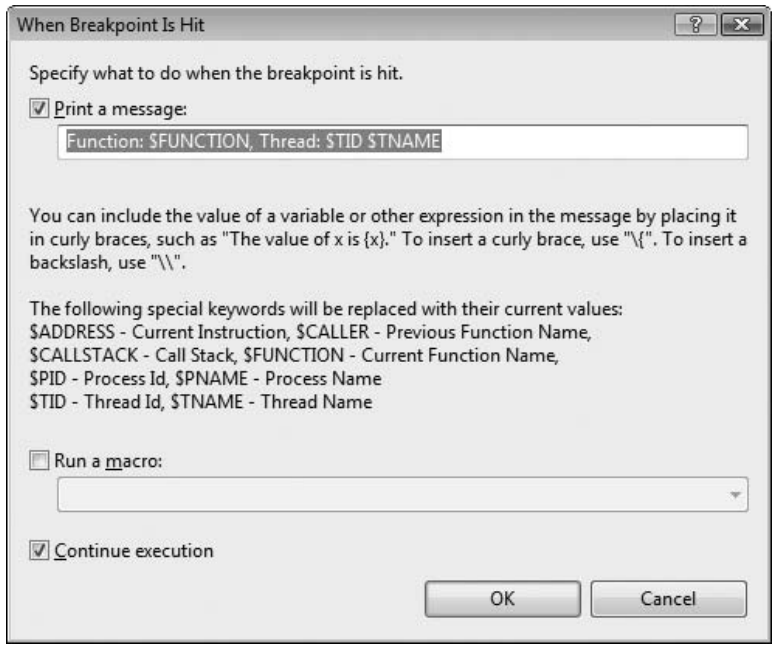

When Breakpoint Is Hit
Specify what to do when the breakpoint is hit.
Print a message:
Function: $FUNCTION, Thread: $TID $TNAME
You can include the value of a variable or other expression in the message by placing it in curly braces, such as "The value of x is {x}." To insert a curly brace, use "\{". To insert a backslash, use "\\".
The following special keywords will be replaced with their current values:
$ADDRESS - Current Instruction, $CALLER - Previous Function Name,
$CALLSTACK - Call Stack, $FUNCTION - Current Function Name,
$PID - Process Id, $PNAME - Process Name
$TID - Thread Id, $TNAME - Thread Name
Run a macro:
Continue execution
OK
Cancel

Figure 8-12. *You can use the When Breakpoint Is Hit dialog box to output a message or run a macro whenever the breakpoint is reached.*

Table 8-3. *Breakpoint glyphs*

Icon	Type	Description
	Enabled	A normal, active breakpoint. If breakpoint conditions or hit count settings are met, execution will pause at this line.
	Disabled	Execution will not pause at this line until the breakpoint is reenabled.
	Error	The location or condition is not valid.
	Warning	The code at this line is not yet loaded, so a breakpoint can't be set. If the code is subsequently loaded, the breakpoint will become enabled.
	Hit Count	A Hit Count condition has been set.

Stepping Through Code

One of the most powerful techniques for debugging an application is to single-step through the code, giving you the opportunity to see the execution flow and to examine the value of variables, properties, objects, and so on. To see this in action, go to the code-behind file in the example. Place a breakpoint on the call to the `Add` method of the `DropDownList` control's `Items` collection, the line in the `Page_Load` method where the items are added to the `DropDownList`. Then, run the program by pressing F5 to start the debugger.

The breakpoint will be hit, and the program will stop execution at the line of code containing the breakpoint, which will turn yellow. The breakpoint glyph in the left margin will have a yellow arrow on top of it. The screen should look like Figure 8-13.

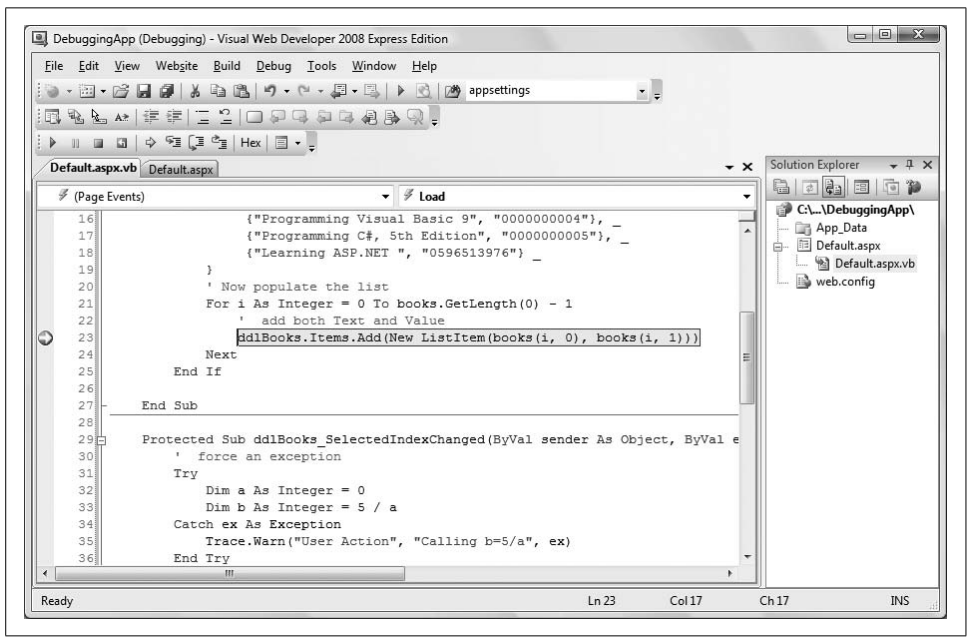


Figure 8-13. *When a breakpoint is hit, the program's execution stops, and the current line of code is highlighted.*

You can now move forward a single statement or line at a time, stepping into any methods or functions as you go, by using one of the following techniques:

- Select the Debug → Step Into menu command.
- Click the Step Into icon (see Table 8-2, shown earlier in this chapter, for a picture of the icon).
- Press F11.

You can step through the code without going through called functions or methods. That is, you can step over the calls rather than into the calls, using one of the following techniques:

- Select the Debug → Step Over menu item.
- Click the Step Over icon (see Table 8-2 for a picture of the icon).
- Press F10.

Finally, if you are debugging in a called method or function, you can step out of that method or function call, using one of the following techniques:

- Select the Debug → Step Out menu command.
- Click the Step Out icon (see Table 8-2, shown earlier in this chapter, for a picture of the icon).
- Press Shift-F11.

Examining Variables and Objects

Once the program is stopped, it is intuitive and easy to examine the value of objects and variables currently in scope. Place the mouse cursor over the top of any variable or object in the code, wait a moment, and then a little pop-up window will appear with its current value.

If the cursor is hovering over a variable, the pop up will contain the type of variable, its value (if relevant), and any other properties it may have.

If the cursor is hovering over some other object, the pop-up window will contain information relevant to its type, including its full namespace, syntax, and a descriptive line of help.

Debug Windows

When the program execution is paused for debugging, a number of windows might appear at the bottom of the IDE, as shown in Figure 8-14. These debug windows are optimized to show program information in a specific way. The following sections describe the most commonly used windows.

All the debug windows can be accessed in one of three ways: with a shortcut key combination, from the Debug → Windows menu command, or from the Windows icon of the Debug toolbar, as indicated in Figure 8-14. Table 8-4 summarizes the debug windows, along with the shortcut keys for accessing each window.

Visual Studio 2008, but not VWD, includes some additional, more arcane debug windows, including ones for Threads, Modules, Registers, and Memory.

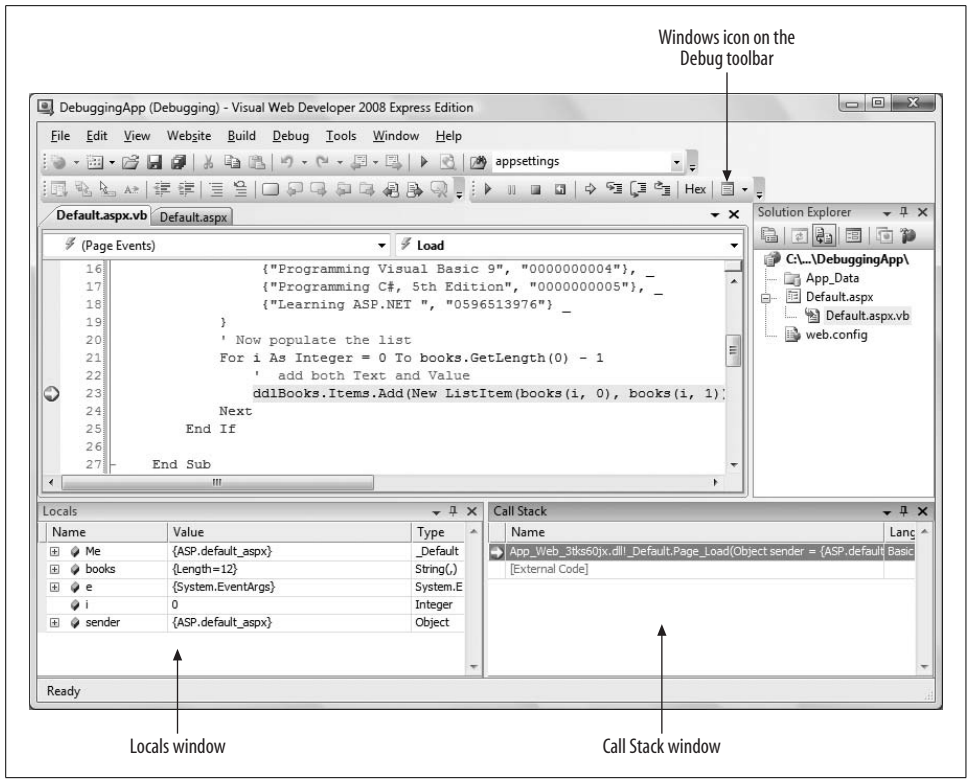


Figure 8-14. *Debug windows are accessible from the Debug menu, a shortcut key combination, or by clicking the Windows icon on the Debug toolbar.*

Table 8-4. *Debug windows*

Window name	Shortcut keys	Description
Immediate	Ctrl-Alt-I	View any variable or expression.
Locals	Ctrl-Alt-V followed by L	View all variables in the current context.
Watch	Ctrl-Alt-W, followed by either 1, 2, 3, or 4	View up to four different sets of variables of your choosing (VWD supports only one).
Call Stack	Ctrl-Alt-C	View all methods on the call stack.

Immediate window

The *Immediate window* allows you to type almost any variable, property, or expression and immediately see its value.

To see the value of an expression, prepend it with a question mark. For instance, if the breakpoint is on the line shown previously in Figure 8-13, you will see the value of the integer `i` by entering the following line:

```
?i
```

in the Immediate window and pressing Enter. Figure 8-15 shows the result of that exercise; additionally, this figure shows the process of assigning a new value to the variable `i` and then viewing its value again. If you change the value of a variable in the Immediate window and then continue to run the program, the new value will now be in effect.

You can clear the contents of the Immediate window by right-clicking anywhere in the window and selecting Clear All. Close the window by clicking the X in the upper-right corner. If you close the window and subsequently bring it back up in the same session, it will still have all the previous contents.

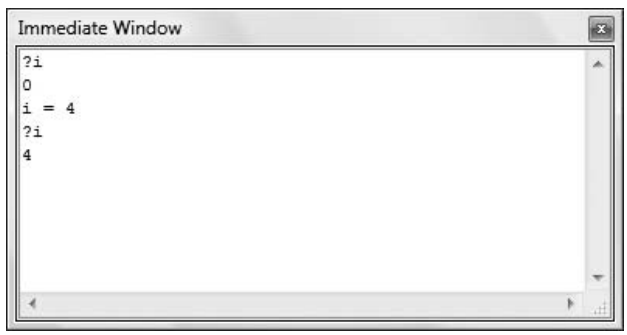


Figure 8-15. *The Immediate window allows you to see the value of a variable at the current breakpoint, and also to change that variable.*

Locals window

The *Locals window* shows all the variables local to the current context displayed in a hierarchical table.

A typical Locals window is shown in Figure 8-16.

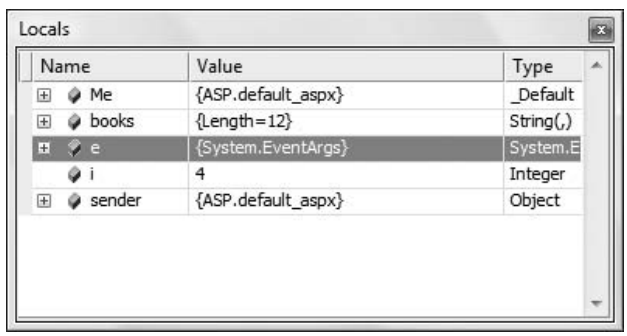


Figure 8-16. *The Locals window shows all the current variables at the point where execution stopped.*

There are columns for the name of the object, its value, and its type. A plus sign next to an object indicates that it has child objects that are not displayed, while a minus sign indicates that its child objects are visible. Clicking a plus symbol drills down the tree and shows any children, while clicking a minus symbol contracts the tree and displays only the parent.

A very useful feature is that values that change in the current step display in red.

You can select and edit the value of any variable. The value will display as red in the Locals window. Any changes to values take effect immediately.

Watch window

The *Watch window* is the same as the Locals window, except that it shows only variables, properties, or expressions you enter into the Name field in the window or drag from another window. The biggest advantage of using a Watch window is that it allows you to decide exactly which objects you want to watch.

In addition to typing in the name of the object you want to watch, you can also drag and drop variables, properties, or expressions from a code window. Select the object in the code you want to put in the Watch window and then drag it to the Name field in the open Watch window.

You can also drag and drop objects from the Locals windows into the Watch window. To do so, both the source window and the Watch window must be open. Highlight a line in the Locals window and drag it down over the Watch tab. The Watch window will come to the foreground. Continue dragging the object to an empty line in the Watch window.

Call Stack window

The *Call Stack window* displays the names of the methods on the call stack and their parameter types and values. You can control which information is displayed in the Call Stack window by right-clicking anywhere in the window and toggling field names that appear in the lower portion of the pop-up menu.

Error Handling

You can and should avoid bugs, but there are runtime errors that cannot be avoided and should be handled as gracefully as possible. You would like to avoid having the end user see ugly or cryptic error messages, or worse, having the application crash. Errors can arise from any number of causes: user action, such as entering invalidly formatted text into a field, program logic errors, or circumstances entirely out of your control, such as an unavailable file or a downed network.

The simplest bugs to find and fix are syntax errors: violations of the rules of the language. For example, suppose you had the following line of code in your VB program:

```
Dim i as Integr
```

When you compile the program, you will get a compiler error because the keyword to declare an integer is misspelled.

Using the IDE dramatically reduces your syntax errors. Depending on how the IDE is configured, any code element that isn't recognized is underlined. If Auto List Members is turned on (Tools → Options → Text Editor → All Languages), the incidence of syntax errors is further reduced. Check the "Show All Settings" checkbox (and leave it checked) to see these options.

If any syntax errors remain or if you are using a different editor, then any syntax errors will be caught by the compiler every time you build the project. It is very difficult for a syntax error to slip by into production code.

When the compiler finds a syntax error, an error message containing the location of the error and a terse explanation will be displayed in the Output window of the IDE. If the error is caused by something such as an unbalanced parenthesis or bracket, or a missing semicolon in C#, then the actual error may not be on the reported line.

More problematic, and often more difficult to catch, are errors in *logic*. The program successfully compiles and may run perfectly well most of the time, yet still contain errors in logic. The very hardest bugs to find are those that occur least often. If you can't reproduce the problem, it is terribly difficult to find it.

While you will try to eliminate all the bugs from your code, you do want your program to react gracefully when a subtle bug or unexpected problem rears its ugly head.

Unhandled Errors

To demonstrate what happens if there is no error handling in place, modify the sample project from this chapter to force some errors.

Go to the code-behind file. Find the `For` loop that populates the `DropDownList` in the `Page_Load` method. Change the test expression to cause an error intentionally at runtime. For example, change the line:

```
For i As Integer = 0 To books.GetLength(0) - 1
```

to:

```
For i As Integer = 0 To books.GetLength(0) + 1
```

When this code runs, it will try to add more items than have been defined in the books array, thus causing a runtime error. This is not a subtle bug, but it serves to demonstrate how the system reacts to runtime errors.

When you run this example in the IDE, execution will stop at the line causing the error, as shown in Figure 8-17, preventing you from seeing the error page. Just press F5 to continue running, or click the Debug → Continue menu item to get to the error page.

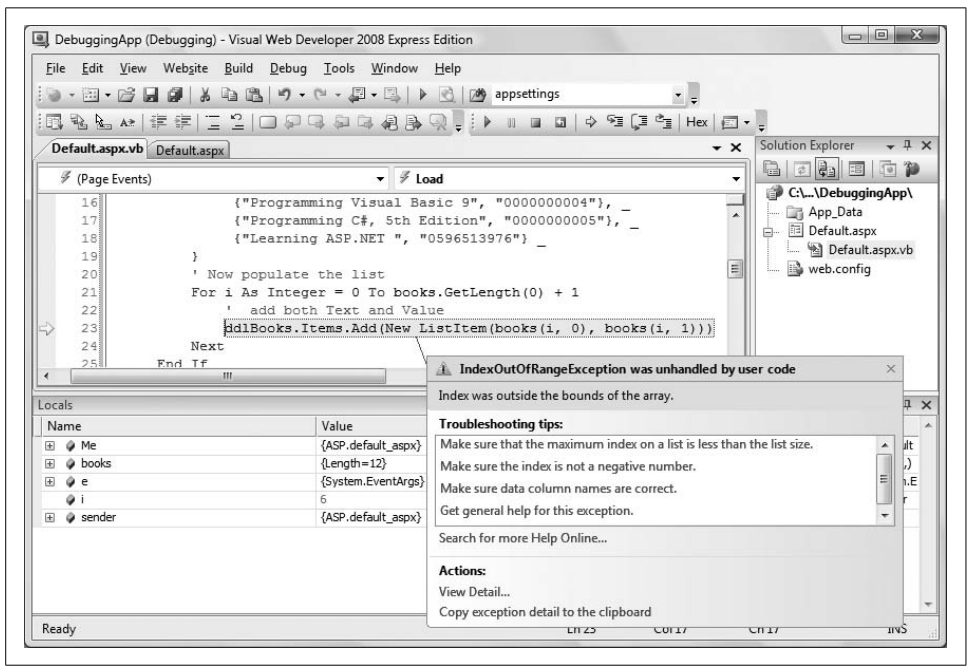


Figure 8-17. *When the program encounters a runtime error, the IDE stops execution and points out the problem.*

Let the program run. As expected, an error is generated immediately, and the generic ASP.NET error page is displayed to the user, as shown in Figure 8-18.

Although scary to most end users, this generic error page is actually fairly useful to the developer or technical support person who will be trying to track down and fix any bugs. It tells you the error type, the line in the code that is the approximate error location, and a stack trace to help in tracking down how that line of code was reached.

You can replace this detailed error page with a custom error page and can control who gets to see what by setting the `mode` attribute of the `CustomErrors` element in the configuration file, as we'll describe next.

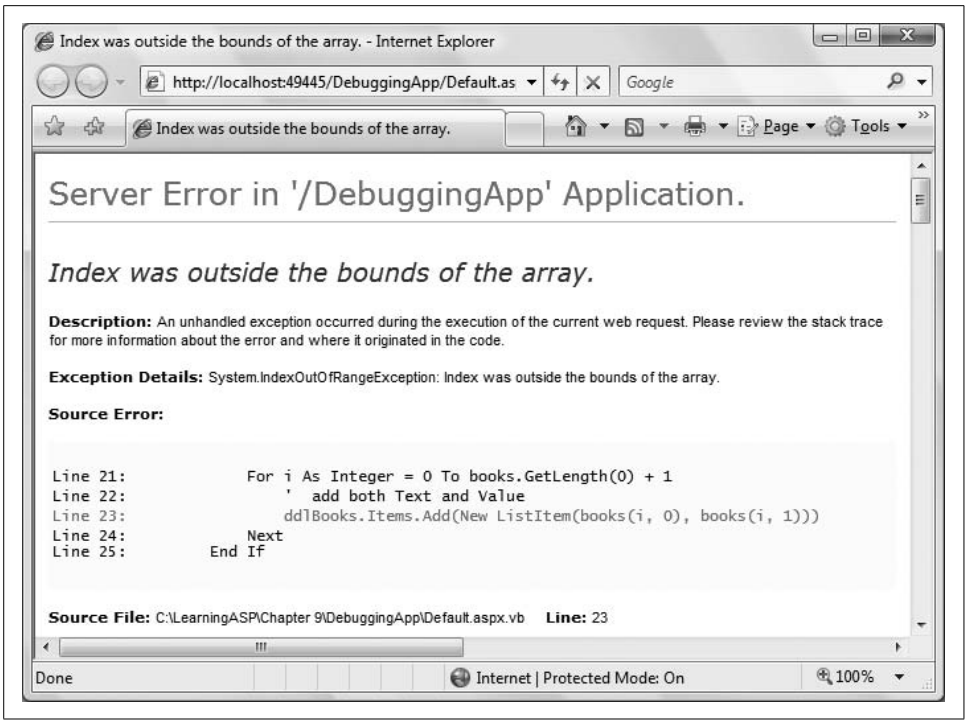

Index was outside the bounds of the array. - Internet Explorer
http://localhost:49445/DebuggingApp/Default.as
Google
Index was outside the bounds of the array.
Page
Tools
Server Error in '/DebuggingApp' Application.
Index was outside the bounds of the array.
Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.
Exception Details: System.IndexOutOfRangeException: Index was outside the bounds of the array.
Source Error:
Line 21: For i As Integer = 0 To books.GetLength(0) + 1
Line 22: ' add both Text and Value
Line 23: ddlBooks.Items.Add(New ListItem(books(i, 0), books(i, 1)))
Line 24: Next
Line 25: End If
Source File: C:\LearningASP\Chapter 9\DebuggingApp\Default.aspx.vb Line: 23
Done
Internet | Protected Mode: On
100%

Figure 8-18. *When you run an application with a logic error, the application may still compile, but fail at runtime. In that case, you'll see a generic error page like this one.*

Application-Wide Error Pages

The previous section showed the default error pages presented for unhandled errors. This is fine for a developer, but if the application is in production, it would be much more aesthetically pleasing if the user were presented with an error page that looked less intimidating.

The goal is to intercept the error before it has a chance to send the generic error page to the client. You do that on an application-wide basis by modifying the configuration file, *web.config*.

The error-handling configuration information in *web.config* is contained within the `<customErrors>` section within the `<system.web>` section, which is contained within the `<configuration>` section. A typical `<customErrors>` section will look like Example 8-5.

Example 8-5. *The custom error section in the web.config file is located in the <system.web> section*

```
<?xml version="1.0"?>
<configuration>
  <system.web>
.
```

Example 8-5. *The custom error section in the web.config file is located in the <system.web> section (continued)*

```
.
.
  <customErrors
     defaultRedirect="CustomErrorPage.htm"
     mode="On"
  />
```

There are two possible attributes for the `<customErrors>` element: `defaultRedirect` and `mode`.

`defaultRedirect` is an attribute that contains the URL of the page to display in the case of any error not otherwise handled. In Example 8-5, the `defaultRedirect` page is `CustomErrorPage.htm`. This example is a simple HTML page contained in the same folder as the rest of the web site. The contents of this page are shown in Example 8-6.

Example 8-6. *The CustomErrorPage.htm file isn't a complex page, but it's a lot nicer to the user than the default error page*

```
<html>
    <head>
       <title>Error Page</title>
    </head>
    <body>
           <h1>Sorry - you've got an error.</h1>
    </body>
</html>
```

If the custom error page to be displayed is not in the same folder as the rest of the application, then you need to include either a relative or a fully qualified URL in the `defaultRedirect` attribute.

`mode` is an attribute that enables or disables custom error pages for the application. It can have three possible values:

`On`
: Enables custom errors for the entire application. ASP.NET will display the custom error page that you specify instead of the default error page.

`Off`
: Disables custom errors for the entire application. ASP.NET will display only its default error pages.

`RemoteOnly`
: Enables custom errors only for remote clients. Local clients (those on the web server) will see the generic error page. In this way, developers can see all the possible error information, but end users will see the custom error page.

Edit your *web.config* file to look like Example 8-5, adding the `customErrors` element inside the `<system.web>` tags; then, select Website → Add New Item, select HTML Page from the list of templates, and add a new item to your web site called *CustomErrorPage.htm*. The full markup for this web page is listed in Example 8-6. Run the program. Instead of Figure 8-18, you will see something like Figure 8-19.

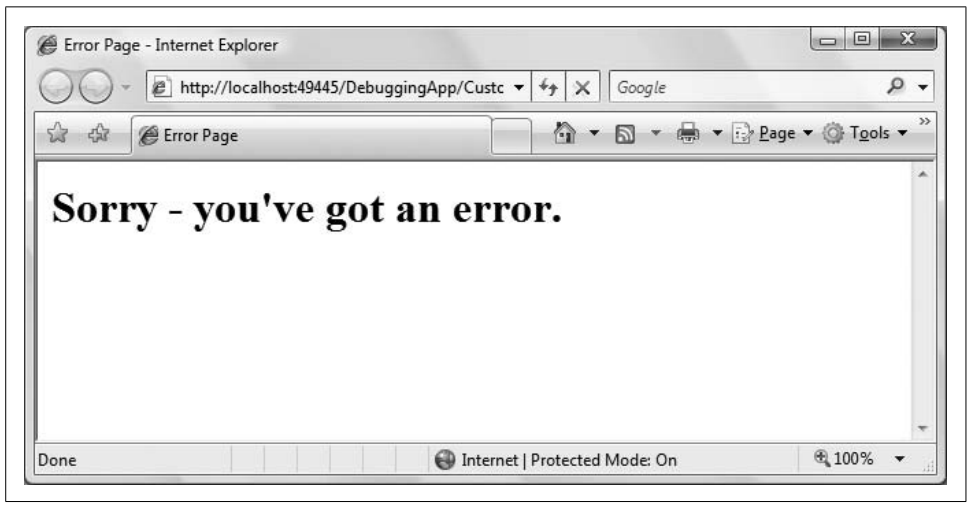


Figure 8-19. *CustomErrorPage resulting from unhandled error with custom errors configured in web.config.*

Obviously, you'll want to put more information on your custom error page, such as instructions or contact information, but you get the idea. Showing dynamic information about the error on the custom error page is also possible.

You can even use a different custom error page for different errors. To do this, you need to include one or more `<error>` child elements in the `<customErrors>` element of *web.config*. For example, modify *web.config* to look like the code snippet in Example 8-7.

Example 8-7. *You can use <error> subtags in web.config to provide customized error pages for specific error codes*

```
<?xml version="1.0"?>
<configuration>

   <system.web>
.
.
.
   <customErrors mode="On"
      defaultRedirect="CustomErrorPage.htm">
      <error statusCode="400" redirect="CustomErrorPage400.htm"/>
```

Example 8-7. *You can use <error> subtags in web.config to provide customized error pages for specific error codes (continued)*

```
      <error statusCode="404" redirect="CustomErrorPage404.htm"/>
      <error statusCode="500" redirect="CustomErrorPage500.htm"/>
  </customErrors>
```

Copy *CustomErrorPage.htm* three times and rename the copies to the filenames in the `<error>` elements in Example 8-7. Do this by right-clicking *CustomErrorPage.htm* in the Solution Explorer and selecting Copy. Then, right-click the web site root folder and select Paste. Next, right-click the new copy of the file and select Rename. Edit the files so each displays a unique message.

Run the program again with the intentional error in the `For` loop still in place. You should see something like Figure 8-20.

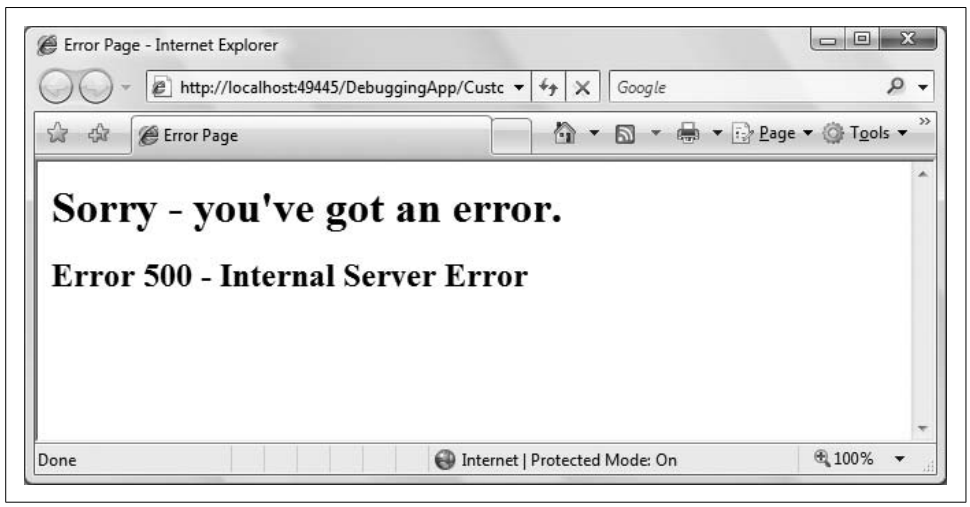


Figure 8-20. *Custom error page for Error 500.*

Fix the error in the `For` loop so the program will at least load correctly. Then, run the program and click the hyperlink you put on the test page. That control is configured to link to a nonexistent *.aspx* file. You should see something like Figure 8-21.

Be aware that you can only display custom error pages for errors generated on *your* server. So, for example, if the hyperlink had been set to a nonexistent page—say, *http://TestPage.comx* (note the intentional misspelling of the extension)—you will not see your custom error page for error 404. Instead, you'll see whatever error page for which the remote server or your browser is configured. Also, you can only trap the 404 error if the page you are trying to link to has an extension of *.aspx*.

Figure 8-21. *Custom error page for Error 404.*

Page-Specific Error Pages

You can override the application-level error pages for any specific page by modifying the `Page` directive at the top of the *.aspx* file.

Modify the `Page` directive in *Default.aspx* file of the DebuggingApp so it appears as follows (note the highlighted `ErrorPage` attribute, which has been added):

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    Inherits="_Default" Trace="false"
    ErrorPage="PageSpecificErrorPage.aspx" %>
```

If there is an error on this page, the `PageSpecificErrorPage.aspx` page will be displayed. If there is an application-level custom error page defined in *web.config*, it will be overridden by the `Page` directive.

Summary

- Tracing allows you to follow the course of your application through the various stages of its life cycle and examine its state as it runs. Tracing appends a great deal of information to the bottom of the page, for which tracing is enabled.
- To turn on tracing for a specific page, add the `Trace` attribute to the `Page` directive of the page you want to trace, and set it to `True`.
- You can insert your own information into the trace log with the `Trace.Write` and `Trace.Warn` methods. They're identical, except that `Trace.Warn` writes in red.

- The `Trace.Write` and `Trace.Warn` methods can accept a message string, a category string and a message string, or a category string, a message string, and an exception object.
- The Visual Studio and Visual Web Developer IDEs provide a complete suite of debugging tools.
- A breakpoint stops execution of an application at a point that you specify to allow you to examine the state of the application at that point.
- When the application is paused at a breakpoint, you can inspect and change the current values of variables and expressions.
- After you've paused at a breakpoint, you can step forward through the application one line at a time, or you can step over or into method calls.
- The Breakpoints window (available only in Visual Studio) shows you all the breakpoints currently in your application.
- You can specify that an individual breakpoint will stop execution if a specific condition is true, or only after being reached a certain number of times.
- You can specify what message is printed when a breakpoint is hit, or you can specify a macro to run when the breakpoint is reached.
- When the application is stopped at a breakpoint, you can simply hover the mouse cursor over objects and variables at that point to see their values and properties.
- The Immediate debug window allows you to type a variable, property, or expression, and see or modify its current value.
- The Locals window shows the variables in the current context and their current values.
- The Watch window is similar to the Locals window, but it shows only those variables that you specify.
- Syntax errors are errors that violate the rules of the programming language. The IDE checks for syntax errors as you write your code, and underlines any code element that doesn't fit. If you attempt to run an application with a syntax error in the debugger, the application will not run, and the IDE will pop up an informational box at the point of the error.
- Logic errors occur when the code is syntactically correct but doesn't behave as intended. These errors are more difficult to deal with because the IDE cannot find them for you. You need to write code to handle those errors so they don't stop your application, or at least provide a way for the application to fail with a minimum of surprise to the user.
- ASP.NET provides default error pages that appear when an unexpected error occurs in the application. These pages have useful information for developers, but not so much for users.

- To prevent the user from seeing the default error pages, you can define custom pages to handle errors.
- When you want to use custom error pages, you have to add a `<customErrors>` section to the *web.config* file and set the `mode` attribute to `On` or `RemoteOnly`.
- You can create specific error pages for individual types of errors. Create a new `<error>` section within the `<customErrors>` section, and specify the `statusCode` attribute for which you want to create a custom page.
- If you want to specify an error page to be used on one specific page of your site, instead of the entire application, add the `ErrorPage` attribute to the `Page` directive, and specify the location of the custom error page.

You've spent the last couple of chapters behind the scenes, figuring out ways to enhance your site and the user's experience that are totally invisible to the user (or should be, if everything goes well). In the next two chapters, it's time to get back to things that the user can see. You'll learn how to change the entire visual appearance of your site with themes and skins, and how to let users choose their preference for viewing your site. Of course, if users are going to customize their experiences, they'll need to identify themselves to you. Or, maybe you don't want just anybody coming into your site and changing things around. That means you'll need some security measures, which is what we'll talk about in the next chapter.

Quiz

1. How do you turn on tracing for a specific page?
2. What is the difference between the `Trace.Write` and `Trace.Warn` methods?
3. What are the three possible arguments to the `Trace.Write` and `Trace.Warn` methods?
4. How do you set a breakpoint in your code?
5. How do you determine the current value of a variable when the application is stopped at a breakpoint?
6. How do you modify the value of a variable while the application is running?
7. What information can you find in the Locals window?
8. What is the difference between syntax and logic errors?
9. What setting do you need to specify before you can use a custom error page?
10. How do you specify that a specific page should use its own custom error page instead of the application-wide pages?

Exercises

Exercise 8-1. Download the file Exercise 8-1 from this book's web site. This application is a part of a page for an online men's clothing store. At the moment, this application runs correctly. Enable tracing on this page, and insert a warning into the trace that indicates when the execution is in the event handler for the drop-down list.

Exercise 8-2. Download the file Exercise 8-2 from this book's web site. This application is similar to the first, but it has a problem. Instead of showing the name of the product in the details pane, some other text is showing up instead, as shown in Figure 8-22. Find the problem and resolve it.

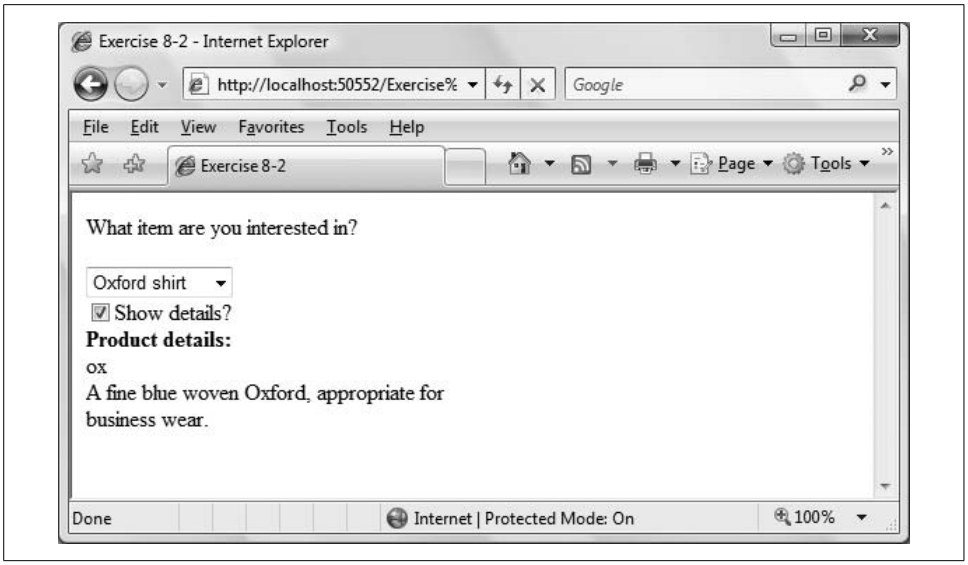


Figure 8-22. *The problem in Exercise 8-2.*

Exercise 8-3. Same site, different problem. Download the file Exercise 8-3 from this book's web site. In this case, the product details don't show up at all, as shown in Figure 8-23. Find the problem and resolve it.

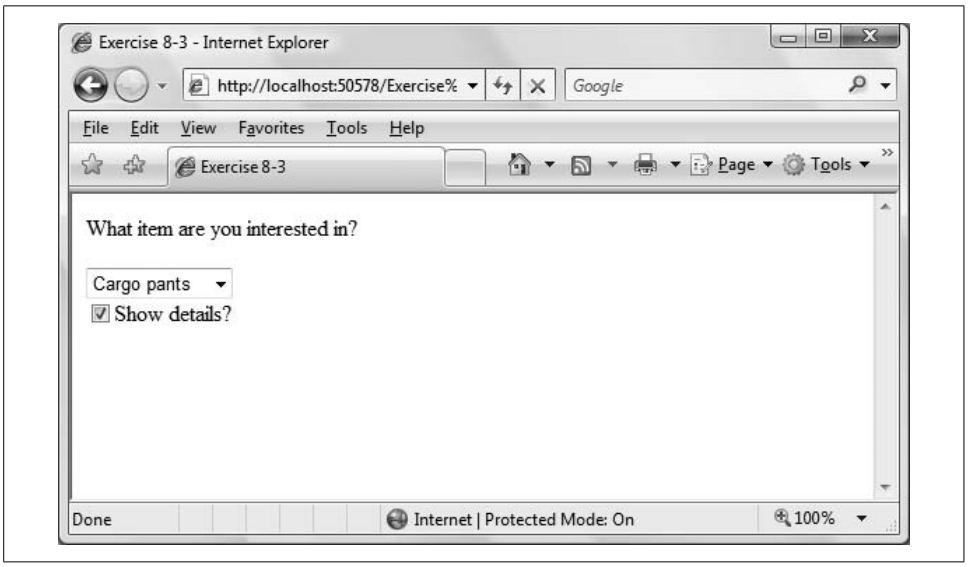


Figure 8-23. *The problem in Exercise 8-3.*

Exercise 8-4. Download the file Exercise 8-4 from the book's web site. The product page now has a hyperlink for users to get customer assistance (and it seems like they'll need it). Unfortunately, the customer assistance page hasn't been created yet. Create a custom error page to handle this error, and give the user the option to return to the product page.

CHAPTER 9

Security

All of the web sites you've created in this book, so far, have allowed any user to go to any page. However, in the real world, you often want to restrict access to your web site. There are many reasons you might want to restrict someone's access to your site: some pages might be free to the public, while others are for registered members only. Or, you might have some parts of your site that only your business partners should access. Or, you might just need to keep out the bad guys. Let's face it, some people get their kicks, or worse, out of wreaking havoc with other people's web sites.

In this chapter, you'll build a functional site with multiple pages, some public and some not. It will include a database of users you will create, and their associated roles. You'll use the ASP.NET login controls, and see just how easy they are to use and customize to your needs. You'll then adapt the site to restrict pages to specific roles, and see how to manage your users. In Chapter 10, you'll see that you can use these same user management tools to allow users to personalize your site to their liking.

Forms-Based Security

Many Internet sites require that users "log in." This allows the site to restrict access to members, and also allows the user to *personalize* the site to their individual needs. These include allowing the site to remember the user's preferences, profile information, shopping cart contents, and so forth, as you'll see in Chapter 10.

Forms-based security is a common technique for validating that the person who is trying to log in to a web site is who they say they are. It presents the user with a web page, or form, containing fields that the user can fill in and submit. Server-side code processes the credentials submitted, such as the username and password, and determines whether the user can proceed.

Not that long ago, creating the "plumbing" of logging in was a tedious and time-consuming job: you had to create a database to track users, create the ability to maintain secure passwords, ensure authorization on each page, assign users to "roles" (such as guest, member, owner, wizard, and more), and so on. You also had to write all the controls to allow the user to log in, to recover passwords, to change passwords, and so forth.

With ASP.NET, all of that has become wonderfully simplified.

Creating Users with the WAT

Visual Studio 2008 offers a wizard called the Web Site Administrative Tool, abbreviated the WAT. This tremendously powerful tool is hidden for no apparent reason under the menu choice Website → ASP.NET Configuration.

If the Web Site Administration Tool used a true acronym, it would be WSAT, but that is very hard to say if English is your first language. Interestingly, the name itself reflects Microsoft's apparent ambivalence (reflected within VS/VWD) of whether "web site" is one word or two. It is used as one word in the menus, but as two in the WAT. We've tried to use it as two words throughout this book, except when we don't.

To try it out, create a new web site called FormsBasedSecurityWAT, and then follow these menu selections. The WAT should open in its own browser window.

Select Website → ASP.NET Configuration. Click the Security tab (or the Security link), and you'll see the page that's shown in Figure 9-1. (Note: "Web Site" is two words in the heading!)

You use this screen for interactively creating users, roles, and access rules. By default, the authentication type is Windows, which means that all user authentication and password management is handled by Windows using your normal Windows sign-on account. This can be quite handy for an *intranet* web site (a web site used only within a single company).

Because we are concerned with Internet accounts (accounts open to the world), change the authentication type to forms-based. To do so, click the "Select authentication type" link under the Users heading, indicated in Figure 9-1.

This brings up the screen shown in Figure 9-2. Select the radio button labeled "From the internet," and then click the Done button.

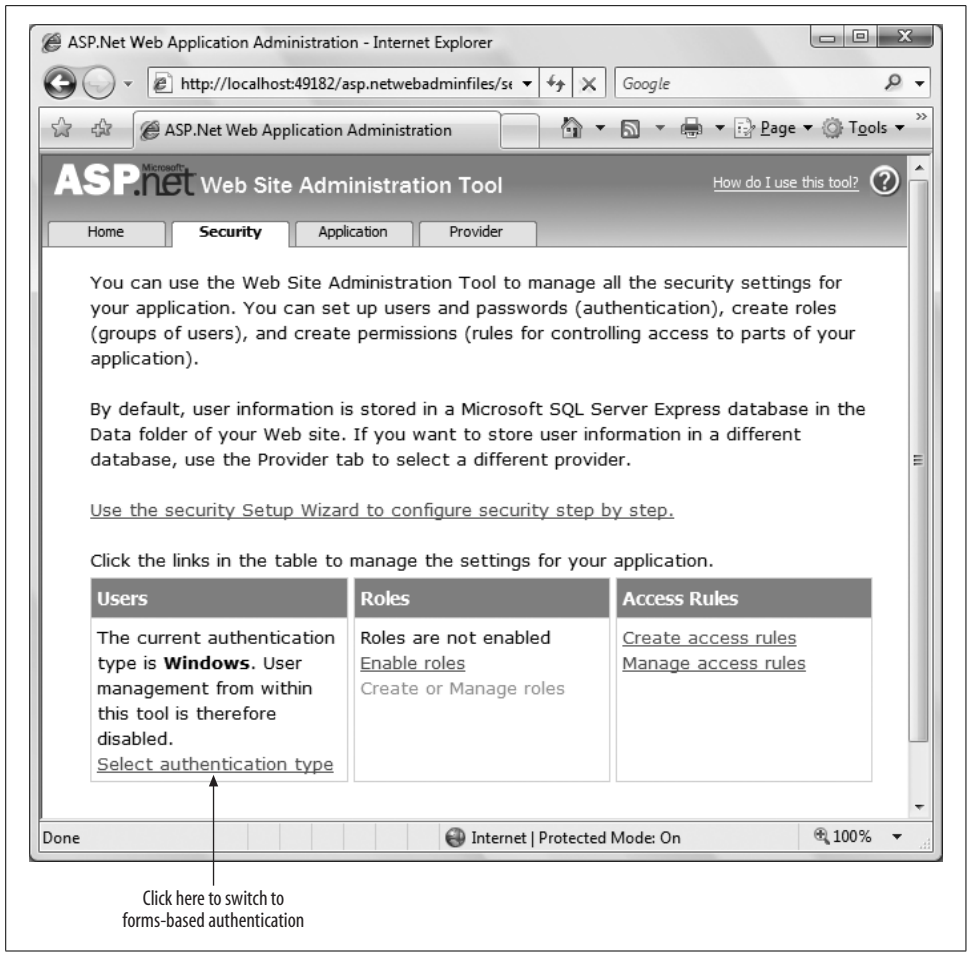

ASP.Net Web Application Administration - Internet Explorer
http://localhost:49182/asp.netwebadminfiles/s
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).
By default, user information is stored in a Microsoft SQL Server Express database in the Data folder of your Web site. If you want to store user information in a different database, use the Provider tab to select a different provider.
Use the security Setup Wizard to configure security step by step.
Click the links in the table to manage the settings for your application.
Users
Roles
Access Rules
The current authentication type is Windows. User management from within this tool is therefore disabled.
Select authentication type
Roles are not enabled
Enable roles
Create or Manage roles
Create access rules
Manage access rules
Done
Internet | Protected Mode: On
100%
Click here to switch to forms-based authentication

Figure 9-1. *The Web Site Administration Tool (WAT) is under the Website menu. On the Security screen, you'll find a link to switch from Windows security to forms-based security.*

Please note the ambiguity of this tool: To get to the WAT, you click "ASP.NET Configuration"; to get forms-based security, you click "From the internet"—you never quite click what you expect, but it does all make sense, sort of.

After you click the Done button, you return to the previous page, but the display under Users has changed, as shown in Figure 9-3. You can create users, manage existing users, or change the authentication type back to Windows.

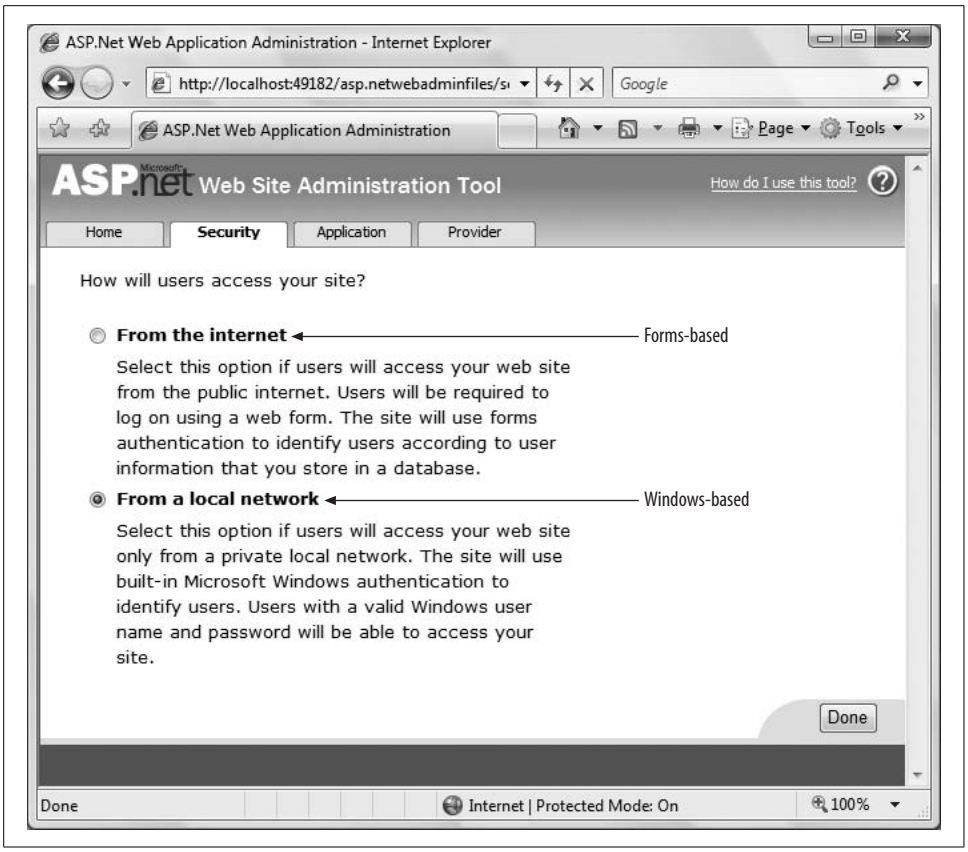

ASP.Net Web Application Administration - Internet Explorer
http://localhost:49182/asp.netwebadminfiles/s
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
How will users access your site?
From the internet
Forms-based
Select this option if users will access your web site from the public internet. Users will be required to log on using a web form. The site will use forms authentication to identify users according to user information that you store in a database.
From a local network
Windows-based
Select this option if users will access your web site only from a private local network. The site will use built-in Microsoft Windows authentication to identify users. Users with a valid Windows user name and password will be able to access your site.
Done
Done
Internet | Protected Mode: On
100%

Figure 9-2. *Select "From the internet" to change the Authentication type from Windows-based security to forms-based security.*

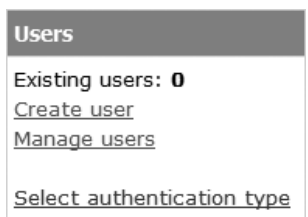


Figure 9-3. *After you change the security type to forms-based, you'll see the number of existing users on the Security tab. You can create new users or manage existing ones from here.*

Click the Create User link to create your first user, as shown in Figure 9-4. Be sure the Active User checkbox remains checked, or else the user will not be able to log into the web site.

Figure 9-4. *Creating a user in the WAT is simple; just enter the pertinent information and click "Create User."*

By default, passwords must be *strong*, which Microsoft defines as: at least seven characters in length and consisting of both alphanumeric and nonalphanumeric characters. So **danh** would not be strong, nor would **danh123**, but **danh123!** would be.

The strong password requirements may differ in other security areas, such as requiring upper-and lowercase letters, or numbers. For more information, go to *http://www.msdn.microsoft.com*, and search for **strong passwords**.

The `CreateUserWizard` control, which you will see later in this chapter, has a `PasswordRegularExpression` property that allows you to substitute your own password requirements. With it, you can dictate how "strong" your passwords need to be.

And, most importantly for your use of the examples in this chapter, remember or write down the passwords you use for all the users you create in this chapter. Otherwise, you will not be able to log in as each of those users as called for in the examples.

Creating at least one user through the WAT sets up a security database with all the tables that ASP.NET's forms-based security system will need to support your use of the forms-based security system.

Your *web.config* file will be modified by the WAT to include the following line under `<system.web>`:

```
<authentication mode="Forms" />
```

In addition, a file-based database will be created in the *App_Data* directory of your application, named *ASPNETDB.MDF*. To view this, click View → Server Explorer to open the Server Explorer Window. If you're using VWD, click View → Database Explorer instead. If you don't already see *ASPNETDB.MDF*, then within that window, right-click Data Connections and select "Add Connection…." This will open the Add Connection dialog box, shown in Figure 9-5.

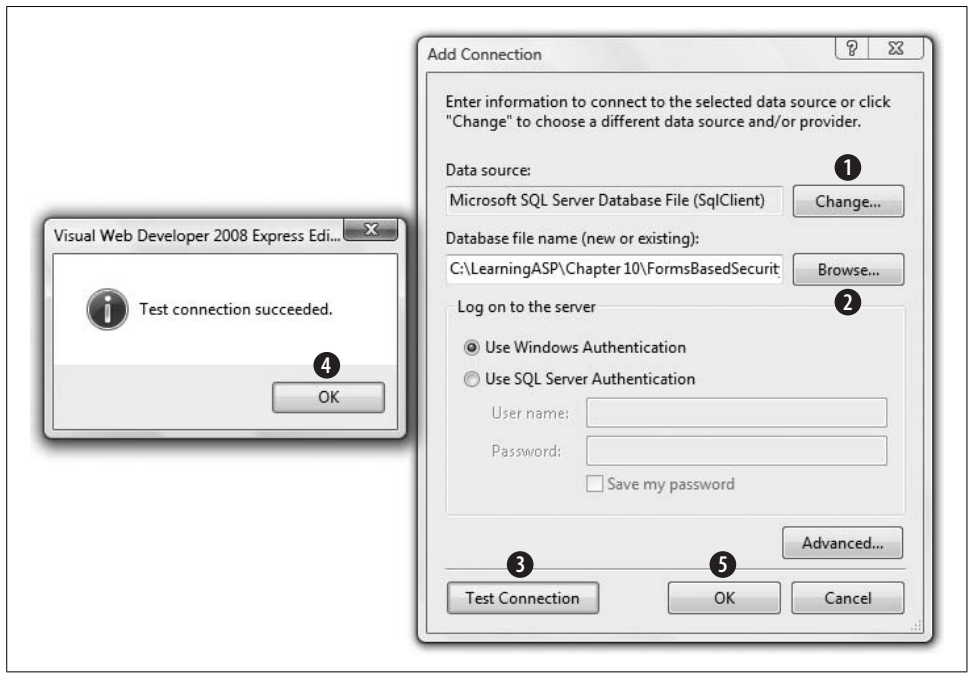


Figure 9-5. *In the Add Connection dialog box, click the buttons in the order shown.*

We've numbered five of the buttons. Let's take them in order.

First, click the button numbered 1 to change the Data Source to the Data File you just created. This will open a new dialog (Change Data Source). From the drop-down, choose Microsoft SQL Server Database File, as shown in Figure 9-6.

Clicking OK will return you to the Add Connection dialog box.

We're up to the button labeled 2. Click the Browse button and navigate to the *App_Data* directory for your new application, and then click the *mdf* file that the WAT created for you.

Leave the radio button set to *Use Windows Authentication*. Don't be confused; you are allowing Windows Authentication to get to the security database only, not to get to the application you are building.

For step 3, click "Test Connection." If all is right, the dialog box (generically named "Visual Web Developer 2008 Express Edition") will open saying that your test connection succeeded. Click the OK button (4), which will return you to the Add Connection button, and you can click OK (5).

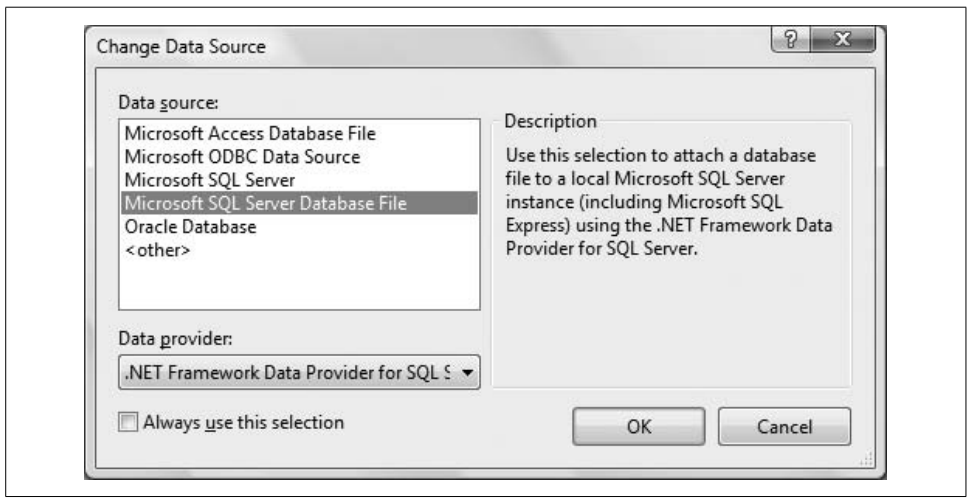

Change Data Source
Data source:
Microsoft Access Database File
Microsoft ODBC Data Source
Microsoft SQL Server
Microsoft SQL Server Database File
Oracle Database
<other>
Description
Use this selection to attach a database file to a local Microsoft SQL Server instance (including Microsoft SQL Express) using the .NET Framework Data Provider for SQL Server.
Data provider:
.NET Framework Data Provider for SQL S
Always use this selection
OK
Cancel

Figure 9-6. *The first step in accessing the Security Database is selecting a Microsoft SQL Server Database File as the Data Source.*

Visual Studio's Server Explorer window (or VWD's Database Explorer) will now show *ASPNETDB.MDF* under Data Connections with a plus sign next to it. Click the plus sign to expand the database, and click the plus sign next to "Tables" to see all the tables in the Security database created for you, as shown in Figure 9-7.

You can look at the data in any of these tables by right-clicking the table name and selecting "Show Table Data."

Managing Users Programmatically

You saw how to create users using the WAT, but that's not the only way to do it. The WAT is useful as a development tool, but it's only usable on a machine that has Visual Studio installed. In almost all cases, this would not be the production web server. Now you will add users programmatically from within your web site. Use the

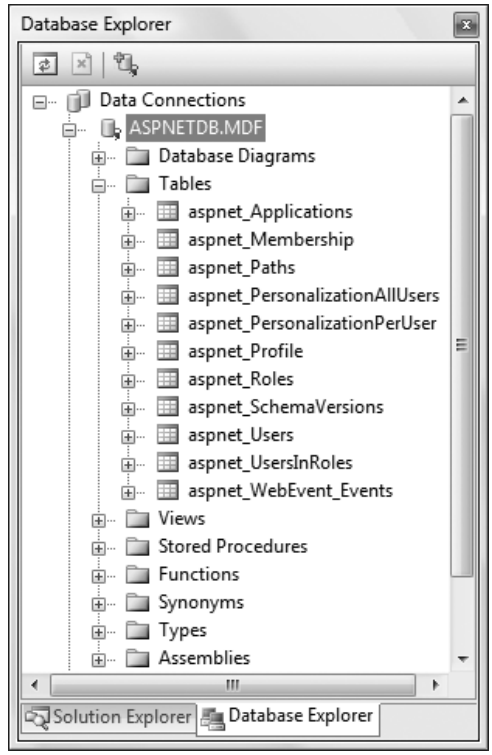


Figure 9-7. *Expanding the tables in the Security Database shows all the tables that the Security Database created automatically.*

web site that you created previously in this chapter, FormsBasedSecurityWAT. You will add two new pages: a welcome page that displays different information depending on whether the user is logged in, and a login page that allows a user to log in.

Creating User Accounts

In a secured web site, before a user can log in, an account for that user must be created. You can do this in the WAT, as you have seen, but accounts can also be created in a live web site.

Look at the web site you created previously in this chapter, FormsBasedSecurityWAT. Delete *Default.aspx* by selecting it in the Solution Explorer and pressing the Delete key.

Make certain that the root of the web site is selected in the Solution Explorer, and add a new page called *CreateAccount.aspx*. Make sure the checkbox for "Place code in separate file" is checked, and the "Select master page" checkbox is not checked.

Depending on what item you have selected in the Solution Explorer when you add a new page, the new web form you add may end up in an unexpected folder. Be careful where you click when you add a new item in the web site.

Switch to Design view, and drag a `CreateUserWizard` control from the Login section of the Toolbox onto the page, as shown in Figure 9-8.

The `CreateUserWizard` control prompts the user for a username, a password (twice), an email address, and a security question and answer—the same information that you provided in the WAT. You can configure what the wizard asks for through the control declaration created in the *.aspx* file for you, through the Properties window, or more commonly, through the Smart Tag, indicated in Figure 9-8.

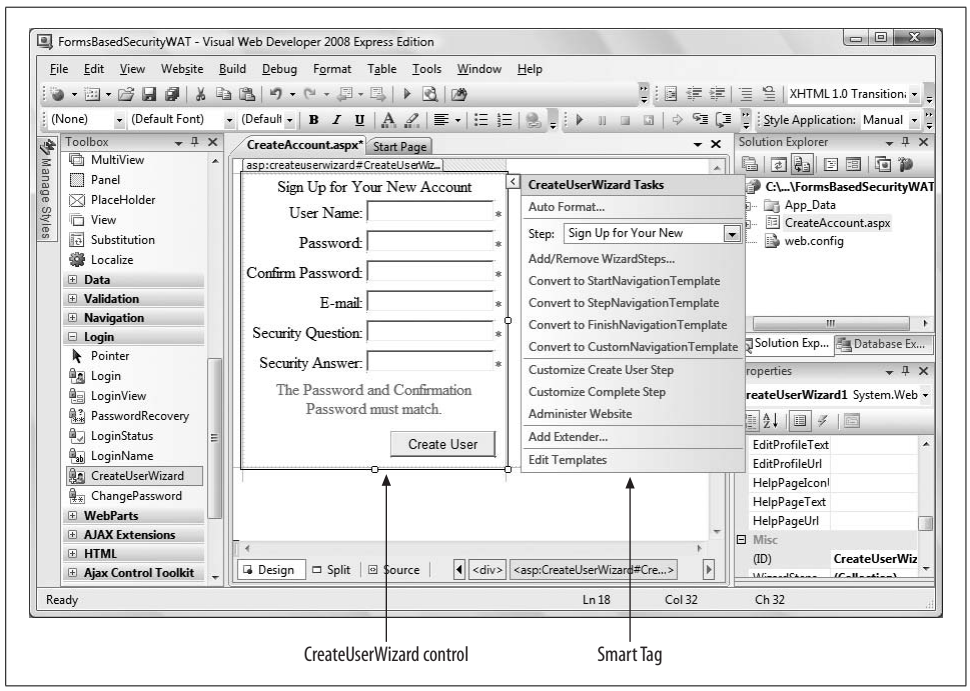


Figure 9-8. *CreateUserWizard control showing the Smart Tag.*

Click the control and find the `ContinueDestinationPageUrl` property in the Properties window. Click in the cell next to the property, then on the ellipses (…) button that appears in that cell. This is the page that the wizard will redirect to after you've created a new user. The Select URL dialog box will appear. Choose *CreateAccount.aspx*—that is, the page itself.

Set the title of the page by clicking the Design window of the page, finding the Title property in the Properties window, and changing it from `Untitled Page` to `Create User`.

Finally, set the *CreateAccount.aspx* as the startup page by right-clicking the page in the Solution Explorer and selecting Set As Start Page. The resulting page is shown in Figure 9-9.

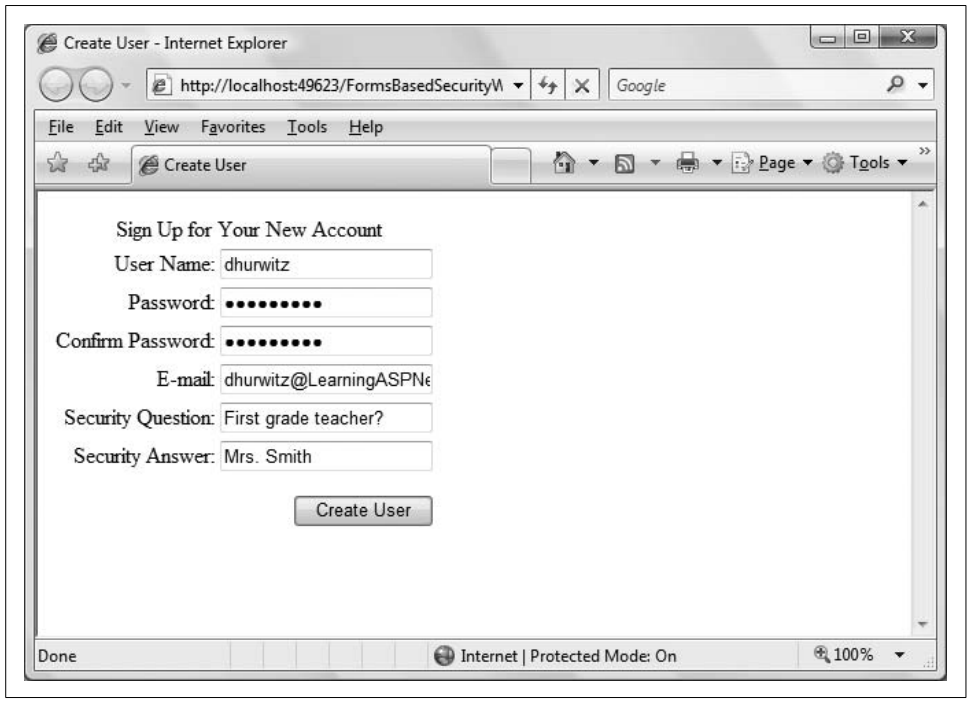


Figure 9-9. *The CreateAccount.aspx page looks complicated, but it's really just one control that handles everything.*

Fill in all the fields (with different values than the user you created before) and click Create User. The new user account will be created, and a confirmation screen will be displayed. Click the Continue button on the confirmation screen, which will bring you right back to the CreateAccount page.

Add a few more accounts, and then stop the application. For these examples, we added the following five users:

bmacdonald
dhurwitz
jliberty
jmontana
tbrady

If you would like, examine the database tables to ensure that the new members have been added.

Creating a Welcome Page

Close the browser if it is open, and then add a new page to the web site called *Welcome.aspx*. Switch to Design view and drag a `LoginStatus` control from the Login section of the Toolbox onto the form.

This control looks like a hyperlink with the text "Login." The Smart Tag for the control indicates that you are looking at the template for when the user is not logged in, as shown in Figure 9-10. You can use the drop-down in the Smart Tag to see the link and text for Logged In status.

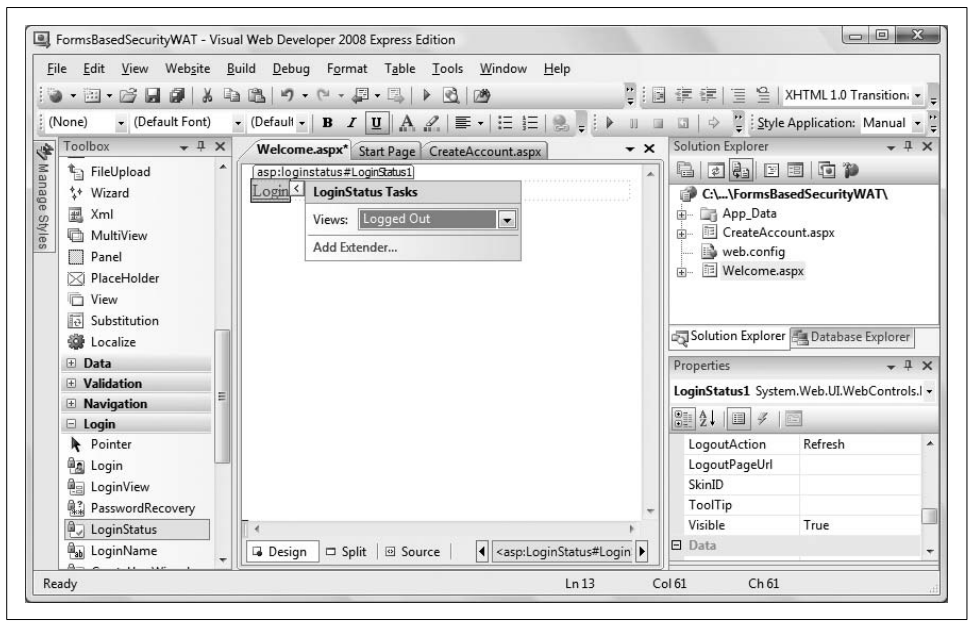


Figure 9-10. *The LoginStatus control has a Smart Tag that you can use to customize the messages for logged-in or logged-out users.*

You can use the Properties window to change the properties of the `LoginStatus` control—for example, to change the displayed text for the logged-in status, `LoginText`, or the logged-out status, `LogoutText`.

It would be nice to see whether you are logged in. To do this, drag a `LoginView` control from the Login section of the Toolbox onto the page. Notice that this control has two views: `AnonymousTemplate` and `LoggedInTemplate`. The template that will be displayed depends on whether the user has logged in—the `AnonymousTemplate` is presented to users who aren't logged in (anonymous users).

Click the Smart Tag and confirm that the view is set to `AnonymousTemplate`. Type some text in the box to display when the user is not logged in, as shown in Figure 9-11.

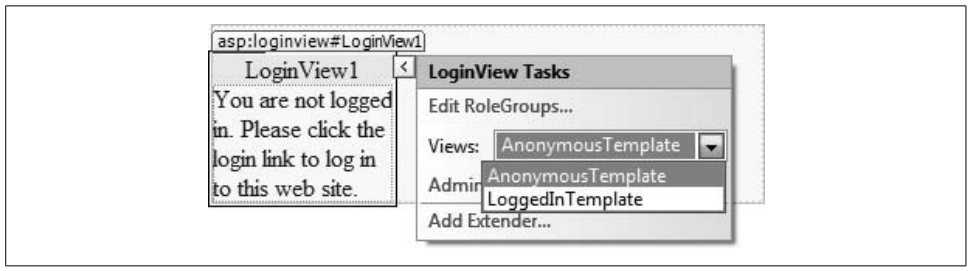


Figure 9-11. *The LoginView control also has two separate templates, depending on whether the user is logged in. Here it's showing the Anonymous template, for logged-out users.*

Click the Smart Tag and select the `LoggedInTemplate`. Drag a `LoginName` control into the box to display the username of the logged-in user along with some text, as shown in Figure 9-12.

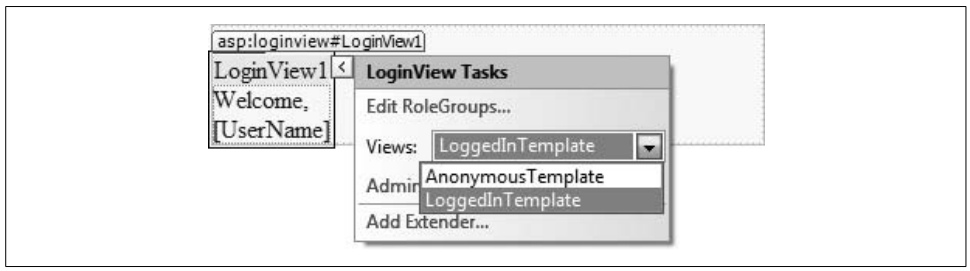


Figure 9-12. *Here the LoginView control is showing the Logged In template, and using a LoginName control to greet the user.*

Creating a Login Page

Add a new page to the web site called *Login.aspx*. It must be called exactly that or the other controls will not know which page to call. Switch to Design view, and drag a `Login` control from the Login section of the Toolbox onto the page, as shown in Figure 9-13. The Login control is the primary control your users will use to log in to your site. To make the page look more professional, click the AutoFormat menu item in the Smart Tag, and pick one of the predefined formats.

Make sure that the Welcome page is the Start page, and then run the application. The Welcome page will display its "Not Logged In" message, as shown in Figure 9-14.

Click the link to go to the login page. Enter the user name and password of one of the users you created previously. If you enter the credentials correctly, you will see a page similar to that shown in Figure 9-15.

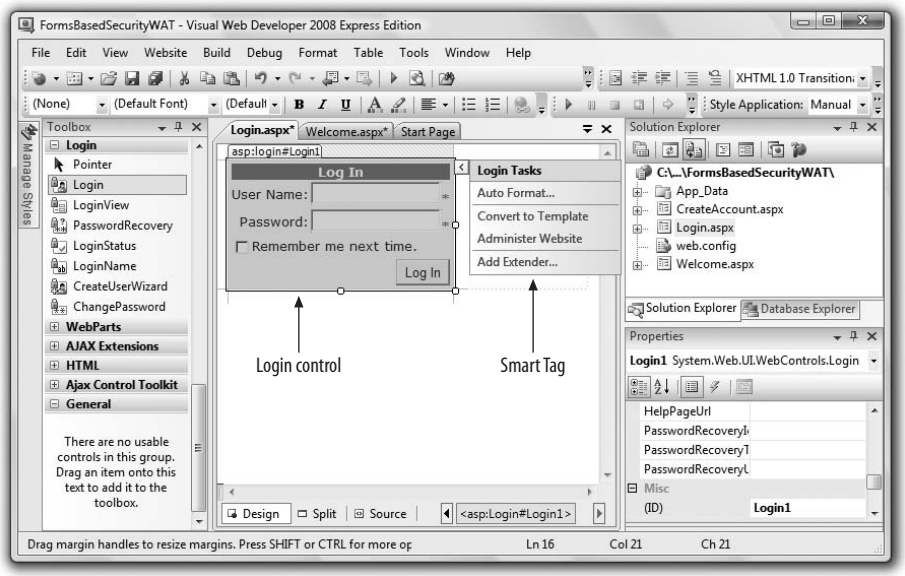


Figure 9-13. *You can use the AutoFormat feature from the Smart Tag of the Login control to give your page a more professional appearance.*

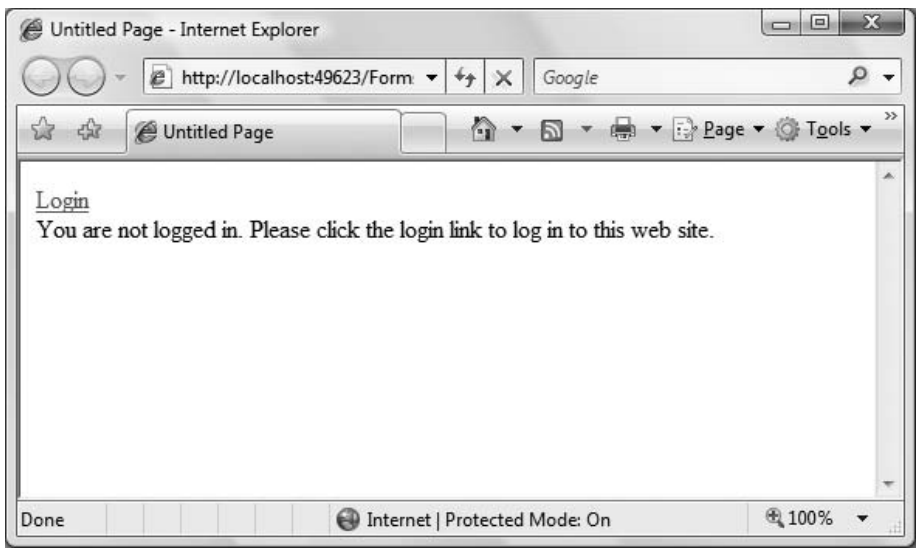


Figure 9-14. *When you first see the Welcome page, you're not logged in, so you're greeted the same as any other user.*

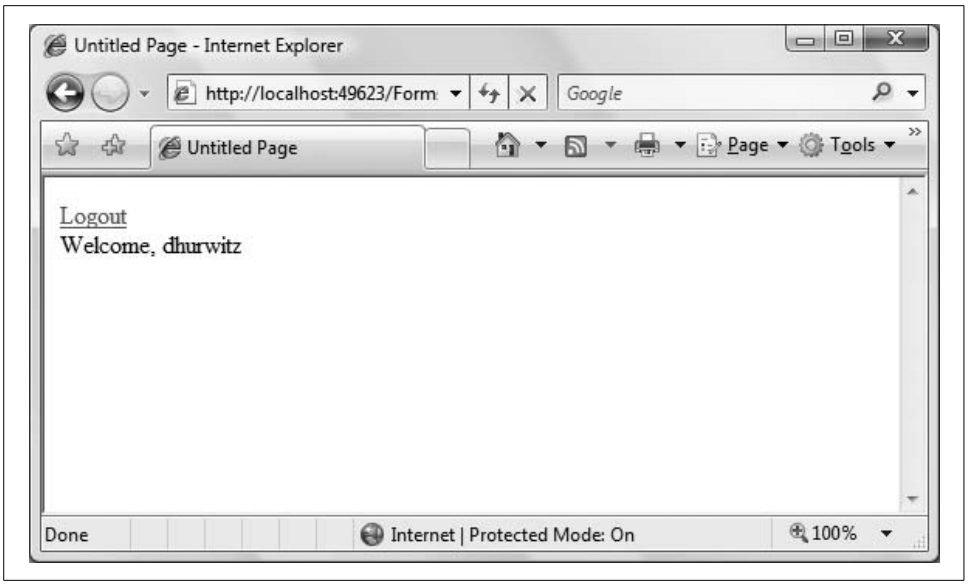


Figure 9-15. *Once you log in successfully, the Welcome Page becomes friendlier.*

If you enter either of the credentials incorrectly, you will see the page shown in Figure 9-16. You can reenter the username and password to try again.

Figure 9-16. *If you don't log in successfully, the Login control provides a message telling you so. This message is generated automatically by the control.*

You can also use a `PasswordRecovery` control, which by default invalidates the current password and sends the user a new one. Using it properly requires you to configure your web server to be able to send email, which is slightly too complex for this example.

Roles

A *role* is a specific set of permissions that has been given a name. Users can be members of one or more roles. For example, a user might be an Administrator, which gives him permission to change data; or a Guest, which does not give him permission to change data. Or, a user can be a member of the Administrator and the User roles, in which case that person will have all the permissions of both roles.

To see this in action, copy the previous example FormBasedSecurityWAT to a new web site—call it SecurityRoles.

Set *Welcome.aspx* as the Start page and run the site to make sure it still works and you can log in.

Use the WAT to enable roles and add the existing users to those roles. Open the WAT by clicking Website → ASP.NET Configuration. When the WAT opens, click the Security tab or link (they are equivalent) to go to the Security page.

There are three management areas across the page, as shown in Figure 9-17. Under the Roles category, there is an "Enable roles" link, indicated in Figure 9-17. Click this link to enable roles.

The link will change to read "Disable roles" and the link below, "Create or Manage roles," will become available. Click that link to create some roles.

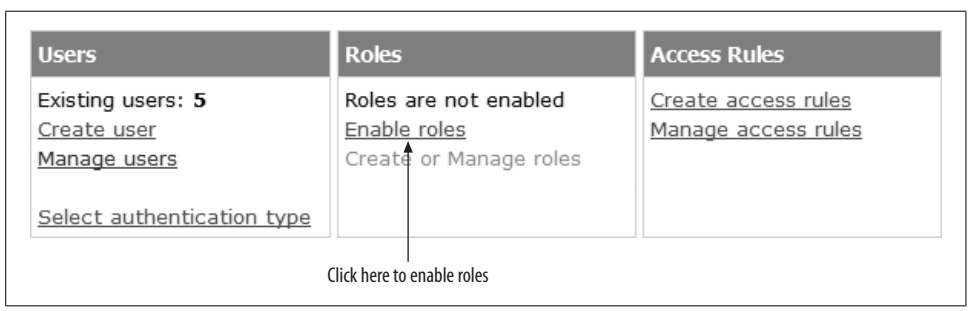


Figure 9-17. *Click the Enable roles link on the Security Page of the Web Site Administration Tool (WAT) to enable roles.*

Enter the name of your first role—`Manager`—in the text box, as shown in Figure 9-18; then, click the Add Role button.

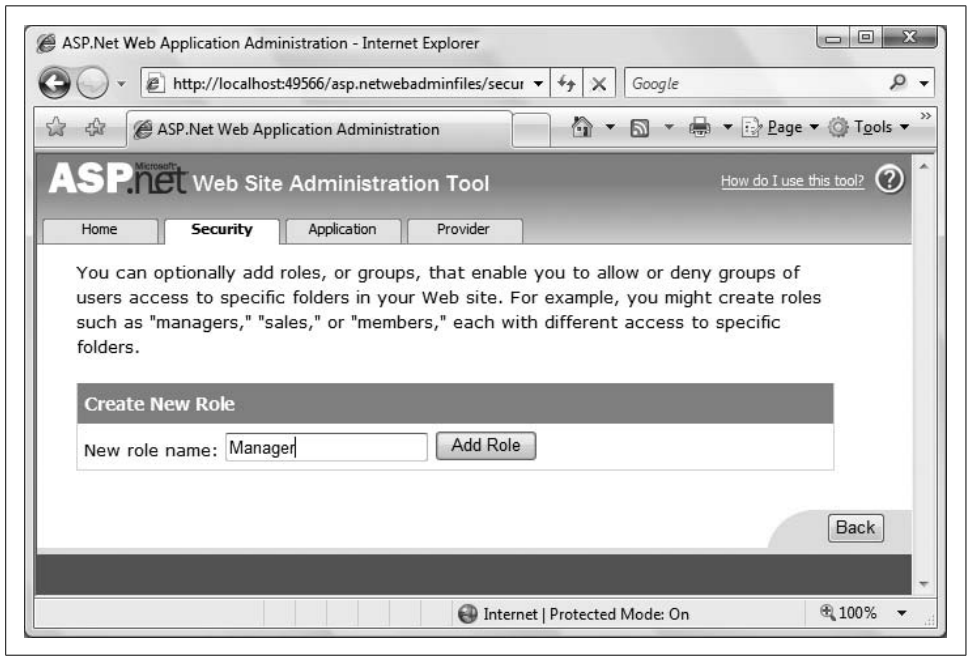

ASP.Net Web Application Administration - Internet Explorer
http://localhost:49566/asp.netwebadminfiles/secur
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
You can optionally add roles, or groups, that enable you to allow or deny groups of users access to specific folders in your Web site. For example, you might create roles such as "managers," "sales," or "members," each with different access to specific folders.
Create New Role
New role name: Manager
Add Role
Back
Internet | Protected Mode: On
100%

Figure 9-18. *Creating a role called Manager in the WAT is simple—just click the Create or Manage Roles link, and create some new roles.*

Add two more roles, `SalesRep` and `Customer`. The screen will now list all of the available roles, as shown in Figure 9-19.

There are links for each role for deleting or otherwise managing the role. The next thing you need to do is add some users to the roles. Under the Add/Remove Users column header, click the Manage link for the Manager role, which brings up a search screen. You can search by username, email address, or, as shown in Figure 9-20, you can click one of the letters to list all the users whose username begins with that letter.

Check User Is In Role for jliberty to be a member of the Manager role, but jmontana is not a member of that role, so do not check that box.

Click the Back button at the bottom-right corner of the screen twice to move back to the Security page, and then click the Manage Users link under the Users category, shown in Figure 9-17, to bring up the user management screen shown in Figure 9-21.

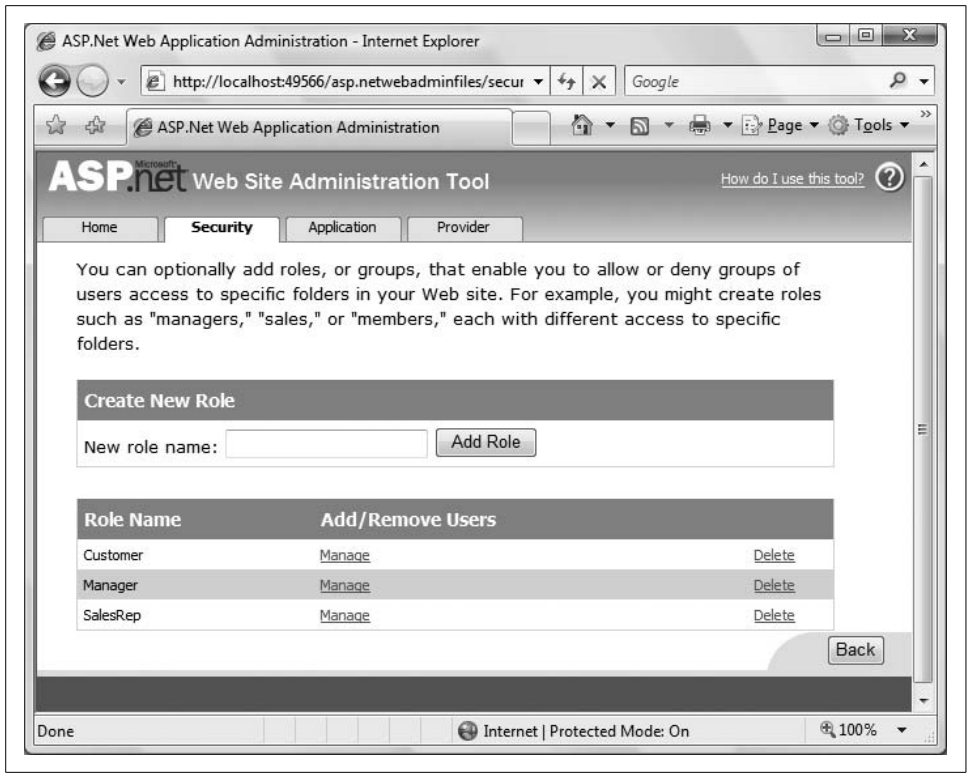


Figure 9-19. *All of the available roles are displayed in the WAT.*

Click the Edit roles link next to the `dhurwitz` name to get a series of checkboxes for adding `dhurwitz` to any of the roles. Add `dhurwitz` to the Manager and SalesRep roles, as shown in Figure 9-22.

While you're on this page, also add `bmacdonald`, `jmontana`, and `tbrady` to the Customer role.

Restricting Access

Although you have set the startup page of the ongoing example to be *Welcome.aspx*, there is nothing to prevent a user, malicious or otherwise, from entering the URL of a specific page, such as *CreateAccount.aspx*, into the address box of a browser.

To see this, run the current example, with *Welcome.aspx* set as the start page, as you have been doing all along. The browser will open *Welcome.aspx*, with an address similar to the following:

http://localhost:1296/SecurityRoles/Welcome.aspx

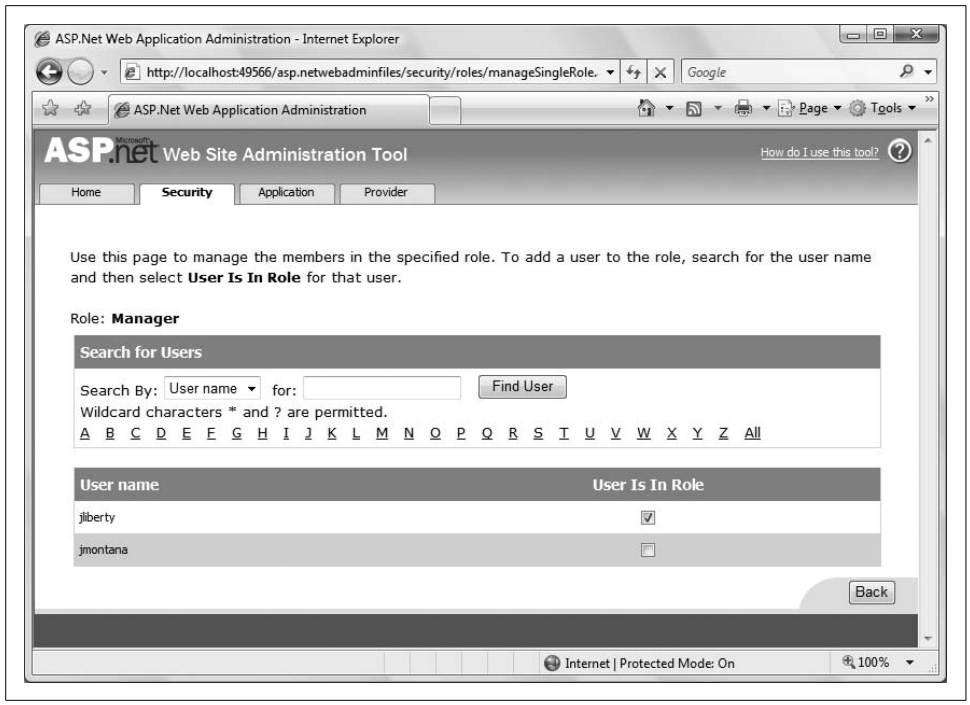

ASP.Net Web Application Administration - Internet Explorer
http://localhost:49566/asp.netwebadminfiles/security/roles/manageSingleRole.
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Use this page to manage the members in the specified role. To add a user to the role, search for the user name and then select User Is In Role for that user.
Role: Manager
Search for Users
Search By: User name for:
Find User
Wildcard characters * and ? are permitted.
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z All
User name
User Is In Role
jliberty
jmontana
Back
Internet | Protected Mode: On
100%

Figure 9-20. *You can easily search for users to add to a given role in the Add/Remove Users page.*

You have probably noticed that every time you run a web site from within the IDE, the URL displayed in the browser address box contains `localhost` and a number separated by a colon. `localhost` refers to the local machine serving a web page to itself. The number is the *port*, or address into a server. Every time the IDE runs a web site, it chooses, at random, a different port to use.

Some port numbers are referred to as *well-known ports*, meaning that they have a standardized usage. For example, port 80 is commonly used for HTTP requests (i.e., web sites), and ports 20 and 21 are commonly used for FTP. The port numbers from 1 through 1024 are reserved for well-known ports.

Edit this address to point instead to the CreateAccount page, as follows:

http://localhost:1296/SecurityRoles/CreateAccount.aspx

That page will open, regardless of your credentials (whether you're logged in and what roles you're in).

This can lead to obvious security problems, providing access to unauthorized users. It can also cause database corruption or exceptions being thrown when your database code assumes that you will have a valid user id and you have none.

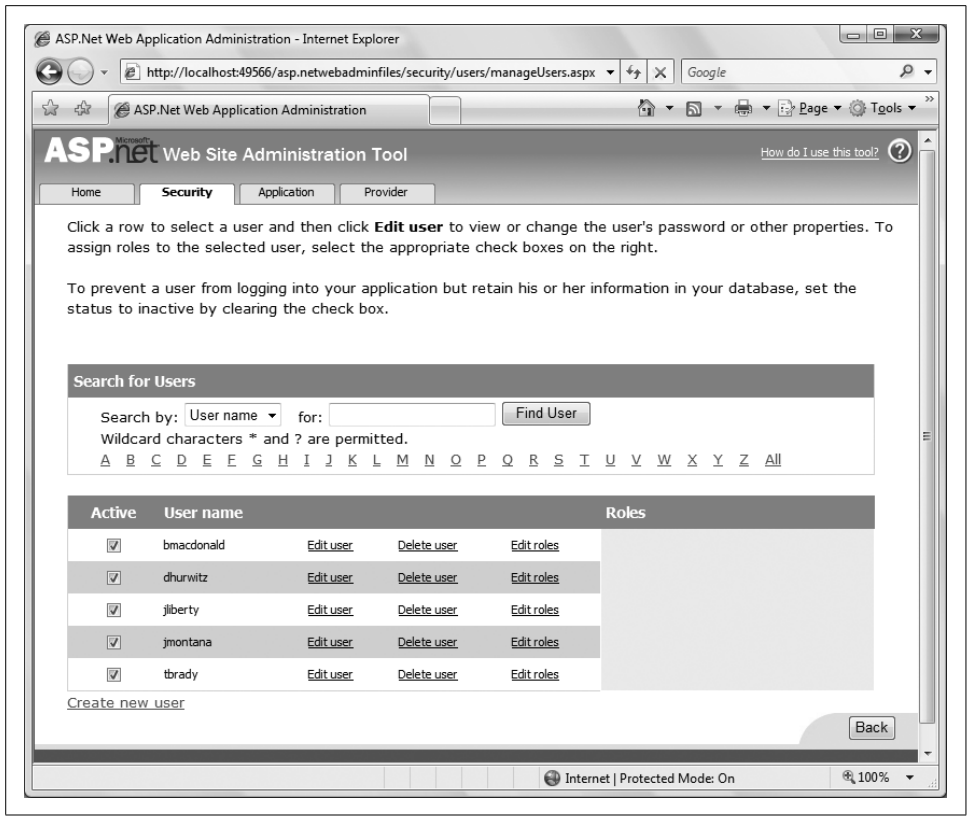

ASP.Net Web Application Administration - Internet Explorer
http://localhost:49566/asp.netwebadminfiles/security/users/manageUsers.aspx
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Click a row to select a user and then click Edit user to view or change the user's password or other properties. To assign roles to the selected user, select the appropriate check boxes on the right.
To prevent a user from logging into your application but retain his or her information in your database, set the status to inactive by clearing the check box.
Search for Users
Search by: User name for: Find User
Wildcard characters * and ? are permitted.
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z All
Active
User name
Roles
bmacdonald Edit user Delete user Edit roles
dhurwitz Edit user Delete user Edit roles
jliberty Edit user Delete user Edit roles
jmontana Edit user Delete user Edit roles
tbrady Edit user Delete user Edit roles
Create new user
Back
Internet | Protected Mode: On
100%

Figure 9-21. *Instead of adding users to roles from the roles page, you can use this screen for managing users in the WAT.*

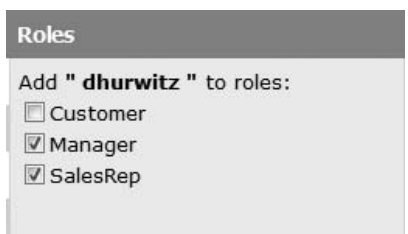


Figure 9-22. *When you click the Edit roles link, the page allows you to add a user to roles.*

To avoid all this, it is good practice to check the login status of the user in the Page Load event of every page in the web site. If the user is not logged in (or is not in the correct role), you can then redirect the user to the appropriate page (often the login page).

This security is not necessary in the normal start page of the web site, *Welcome.aspx* in the example, because the `LoginStatus` control on the page already takes care of this. It may also not be necessary in many "open" pages on public sites where you may not want to force visitors to log in until it is absolutely necessary (to retrieve their own personalized data or to place an order).

Testing for Login Status

To see how checking of credentials might be accomplished, open the code-behind file for CreateAccount, *CreateAccount.aspx.vb*. In the Class Name drop-down at the top left of the editing window, select (`Page Events`), and in the Method Name drop-down at the top right of the editing window, select the `Load` event. This will insert an empty code skeleton for the `Page_Load` event handler.

Type the following code inside the `Page_Load` method:

```
If User.Identity.IsAuthenticated = False Then
   Response.Redirect("Login.aspx")
End If
```

Now run the application. Before logging in, edit the page address in the browser to go to *CreateAccount.aspx*. Instead of *CreateAccount.aspx* opening, you will be immediately taken to the login page.

If you enter a valid username and password at this point, the Login control will try to redirect to *Default.aspx*, a page that does not exist. Instead, set the `DestinationPageUrl` property of the `Login` control to one of the pages in the web site, such as *Welcome.aspx*. Then, on a successful login, the user will be redirected to that page.

On the other hand, if you run the web site and log in, and then edit the browser address to open *CreateAccount.aspx*, you will in fact go to that page.

Testing for Role-Based Authentication Membership

You can also limit access to pages based on the role, or roles, to which the current, logged-in user belongs.

Add another page to the SecurityRoles web site called *ManagersPage.aspx*. As the name implies, this page will be accessible only to managers. To keep the example simple, for now this page will have only an identifying heading and a button to return to the Welcome page, shown in bold in Example 9-1. You can do this in either Source or Design view.

Example 9-1. *The markup for the ManagersPage isn't particularly important; the key is in the code-behind*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="ManagersPage.aspx.vb"
Inherits="ManagersPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>

    <form id="form1" runat="server">
    <div>
        <h1>Manager's Page</h1>
        <asp:Button ID="btnWelcome" runat="server"
            Text="Return to Welcome" />
    </div>
    </form>
</body>
</html>
```

Switch to Design view and double-click the Return to Welcome button to open up an event handler for `Click` event. Add the following highlighted line of code:

```
    Protected Sub btnWelcome_Click(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles btnWelcome.Click
        Response.Redirect("Welcome.aspx")
    End Sub
```

While you are at it, add a button to the Welcome page for navigating to the Manager's Page, as shown in Figure 9-23. Set the ID of the button to `btnManagersPage`, because you will be referring to the button in code elsewhere, and set its Enabled property to `False`. In a moment, you will add some code to the `Page_Load` event handler to enable or disable the button depending on the login status.

Double-click that button in Design view, and add the following highlighted line of code to the `Click` event handler:

```
    Protected Sub Button1_Click(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles btnManagersPage.Click
        Response.Redirect("ManagersPage.aspx")
    End Sub
```

Next, add an event handler for the `Page_Load` event. Then add the following highlighted code to run every time the page loads:

```
    Protected Sub Page_Load(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles Me.Load
        If User.Identity.IsAuthenticated = True Then
            btnManagersPage.Enabled = True
        Else
```

```
            btnManagersPage.Enabled = False
        End If
    End Sub
```

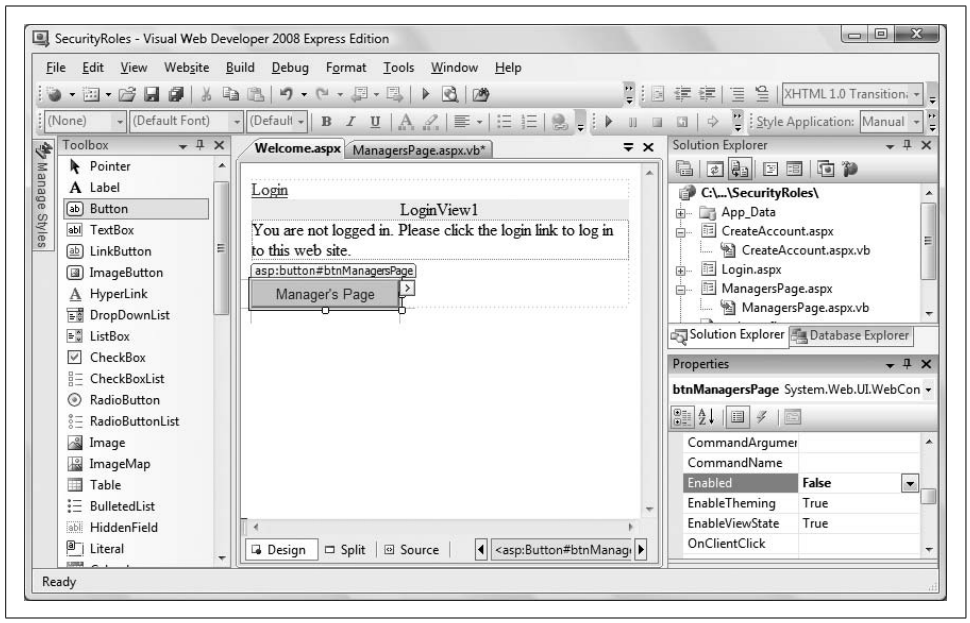


Figure 9-23. *Add a button to the Welcome page in Design view, for navigating to the Manager's page.*

Including an `If` and an `Else` clause ensures that the `Enabled` state of the button will always be what you want, regardless of the circumstances.

Run the app. The Welcome page opens with the Login link and contents of the AnonymousTemplate displayed, and the Manager's Page button is disabled.

Log in as a user in the Managers role, say `dhurwitz`, and the Manager's Page button will become enabled. Click that button to move to the Manager's Page; then, click the button on that page to return to the Welcome page.

There is still a problem with this application, however: if you log in with one of the usernames that are not in the Manager's role, such as `tbrady`, you still are allowed to go to the Manager's page. Let's fix this.

Go to the code-behind for the Manager's page, *ManagersPage.aspx.vb*. Create an event handler for the Page Load event. Enter the following highlighted code to the event handler:

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    If User.IsInRole("Manager") = False Then
        Response.Redirect("NoPrivs.aspx")
    End If
End Sub
```

This code will redirect to a page called *NoPrivs.aspx* if the current user is not a member of the Manager role. So, create that page, making it very similar to *ManagersPage.aspx*, with only a heading, some text, and a button to redirect back to the Welcome page, as shown in Figure 9-24.

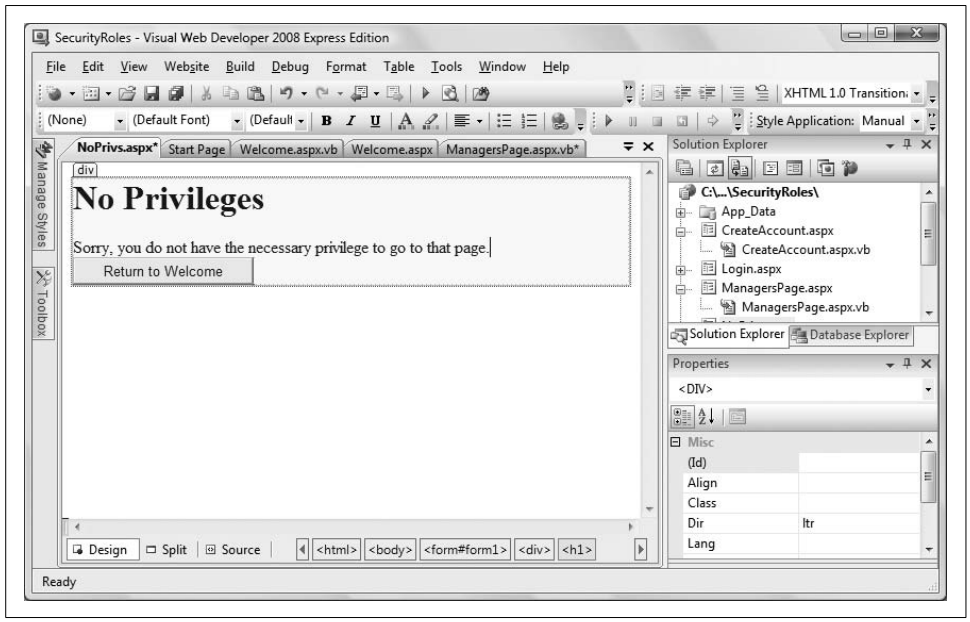


Figure 9-24. *The NoPrivs page in Design view. This is the page where users will be redirected if they try to access a page for which they do not have permission.*

Double-click the button to open an event handler for the `Click` event and enter the following highlighted line of code:

```
Protected Sub btnWelcome_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles btnWelcome.Click
    Response.Redirect("Welcome.aspx")
End Sub
```

Now, when you run the app, if you log in with a username that is a member of the Manager role, you can navigate to the Manager's page. Otherwise, you cannot get to the Manager's page; you are directed instead to NoPrivs.

In a real application, it would make more sense to only enable the Manager's page button on the Welcome page if the user were a member of the Manager role, not simply logged in. However, then we would not have an easy way to demonstrate how to restrict access to a page based on the role of the current user.

There's a lot more to properly securing a web site than we have space to talk about here. This chapter is really just the tip of the iceberg; for a more detailed discussion, see *Programming ASP.NET* (O'Reilly).

Summary

- ASP.NET provides controls you can use to allow a user to log in to your site.
- Forms-based authentication is a technique for validating the identity of the user of the web page, and is useful for users connecting across the Internet.
- Windows-based authentication lets Windows handle the authentication tasks, and is most useful for users connecting on an intranet.
- Before users can log in to your site, you need to create user accounts for them. One way to do this is administratively, using the Web Site Administrative Tool (WAT). The WAT presents itself in a separate browser window, and allows you to choose between Windows- and forms-based authentication.
- To create a new user with the WAT, you simply need to select the link in the WAT, and fill in the user's name, password, email address, and a security question. The information will automatically be added to your site's security database.
- Instead of using the WAT, you can also create user accounts and add them to your security database from within your site. ASP.NET provides a `CreateUserWizard` control that lets users create their own accounts when your site is running. Users provide all the same information, and the account is automatically created.
- The `LoginStatus` control has two views: one for logged-in users, and one for users who aren't logged in yet. Logged-in users are greeted with a message you can configure, and other users are given a link so they can log in.
- The `LoginView` control also has two templates for logged-in and not-logged-in (anonymous) users. You can use this control to display customized content based on the user's login status.
- The `Login` control is the control where users enter a username and password to log in to your site.
- The `PasswordRecovery` control automatically resets a user's password and sends them an email notifying them of the change. This requires an SMTP server.
- Another way to determine what users can and can't see on your site is to use roles. You define a specific role, assign users to that role, and then specify what pages members of the role can or cannot access.
- You can enable roles, create new roles, and add users to roles from the WAT.
- A good practice on sites where security is important is to check the login status of the user in the Page Load event of every page.
- The `User.Identity.IsAuthenticated` property is a Boolean that you can use to take action based on whether the user is logged in.
- To determine if a user is a member of a specific role, you can use the `User.IsInRole` property.

As you've seen, security is very important for keeping unwanted users out of sensitive areas of your site. The combination of the WAT and ASP.NET `Login` controls make it very easy for you to define who gets to access which page. The benefits go beyond security, though. When users are logged in, you can save data about them that they provide, allowing you to greet them by name when they visit, keep track of their shopping cart, and even customize the appearance of your site to fit their preferences. Giving users some degree of control over your site builds a stronger relationship between you and them, and that keeps them coming back. That's called *personalization*, and you'll see how to do it in the next chapter.

Quiz

1. What are the two methods for creating users for your site?
2. What is the difference between forms-based security and Windows authentication?
3. What is the purpose of the security question in the user creation process?
4. Where is the user information stored in your site?
5. What control do you need to add to enable users to create their own accounts?
6. How many views does the `LoginStatus` control have?
7. What's the purpose of the `LoginName` control?
8. What does the `Login` control do if the user enters an incorrect password?
9. What tool do you use to add users to roles?
10. What property do you use to restrict access to a page based on a role?

Exercises

Exercise 9-1. In this set of exercises, you're going to create a web site for Ajax Security, a company that provides security guards to banks, jewelry stores, and other clients. The company employs several guards, each of which works the night shift or the day shift, but not both. Create four user accounts for the guards, and assign them to the appropriate shifts, as follows:

Day shift: anakamura, wsmith
Night shift: jsharma, toliver

Create a web site with three pages: a welcome page, a login page, and a page with the schedules for the guards—one schedule for the day shift, and one for the night shift. (You don't actually have to provide a schedule; a placeholder will do.) For security's sake, guards working the day shift shouldn't be able to see the night shift schedule, and vice versa.

Exercise 9-2. You want potential customers to be able to use the site, and to be able to log in so that they can access customer-specific information. However, you don't want any logged-in customer to be able to access the guard schedules. Add a "client information" page to the site, with placeholder content, and a "create an account" page to the site so that users can create their own accounts. Once they've created an account, users should be able to access the client information page, but not the guards' schedules. Add a page to redirect an unauthorized user who tries to access the guards' page.

Exercise 9-3. The managers of the company need access to all the existing information, plus a page that neither the guards nor the clients can access. Add a "managers" page to the site, with placeholder content. The managers shouldn't be able to create their own logins; they'll have to be created by the site administrator (that's you). Add two managers: mrand and dknight. The managers should have access to the manager page, but should also be able to see the schedules for all the guards. Be sure to redirect an unauthorized user who tries to access the managers' page.

CHAPTER 10

Personalization

As your site grows, and you have lots of information available, you'll find that many users will only want to make use of a subset of what you have available. Say you have a site that offers news articles from around the world. Some users might only want to see news from their home countries; others might want only financial news; others might want just the headlines. In short, you want your site to display different things to different users, based on the user's preferences. You can do that with ASP.NET, and it's called *personalization*.

Personalization allows users to modify a site's settings to reflect their own tastes. It also allows you to keep track of users' "progress" through a sequence of steps or selections made on a page from one visit to another. Many sites use personalization to create persistent wish lists, shopping carts, and so forth. This used to be a huge and complicated job—keeping track of a user's set of preferences and the state of a user's personal information. Fortunately, that is all made easier now with ASP.NET.

In this chapter, you'll build on the project from the previous chapter because the mechanisms for security and personalization both make use of user logins. You'll enhance your site by letting users enter personal information that you'll store and produce on demand, and you'll learn how to provide content for users who'd rather remain anonymous. Finally, you'll modify the appearance of the controls on your site with themes, and you'll see how you can let users set their own themes and retain those settings with their other personal information.

Profiles

Copy the example, SecurityRoles, from Chapter 9 to a new web site called SitePersonalization. Set *Welcome.aspx* as the start page and run it to make sure everything still works.

One of the easiest ways to personalize a site is to define profile information that you will maintain for each user. Profile information can include such simple data as the

user's real name, address, and telephone numbers, or, as you'll see later, it can include more complex developer-defined data. Don't use profile data for sensitive user information; use a secure database instead.

Simple Data Types

To use profiles, you have to make some modifications to your *web.config* file for the project. The first step is adding a new section called `<profile>`, setting the `enabled` property to `true`, and specifying `defaultProvider` as `AspNetSqlProfileProvider`. The `defaultProvider` holds the data for the profiles; `AspNetSqlProfileProvider` is the built-in provider for ASP.NET.

Then, you need to modify *web.config* to indicate which pieces of profile information to save. You add a `<properties>` section to the `<profile>` declaration, which is inside the `<system.web>` section, and then use the `<add>` attribute to add the names of any profile data you want to save. Add the highlighted lines in Example 10-1 to your *web.config*.

Example 10-1. *You need to make some modifications to web.config to enable profiles (with much of the file omitted for brevity)*

```
<?xml version="1.0"?>
<configuration>
   <configSections>...</configSections>
   <appSettings/>
   <connectionStrings/>
   <system.web>
      <roleManager enabled="true" />
      <compilation debug="true" strict="false" explicit="true">...</compilation>
      <profile enabled="true" defaultProvider="AspNetSqlProfileProvider">
         <properties>
            <add name="lastName"/>
            <add name="firstName"/>
            <add name="phoneNumber"/>
            <add name="birthDate" type="System.DateTime"/>
         </properties>
      </profile>
      <pages>...</pages>

      <!-- stuff omitted for brevity  -->

   </system.web>
</configuration>
```

Your *web.config* file may look somewhat different depending on your machine configuration and the databases you have installed. Also, boilerplate comments and lines unrelated to this topic have been removed from Example 10-1.

The configuration shown in Example 10-1 causes ASP.NET to create storage for four pieces of information: first and last name, phone number, and birth date. The default storage type is `String`. Notice, however, that you are storing the birth date as a `DateTime` object.

You can gather this personalization information any way you like. For this example, open *Welcome.aspx* and switch to Design view. Click the Smart Tag of the LoginView control and select the LoggedInTemplate view, as shown in Figure 10-1.

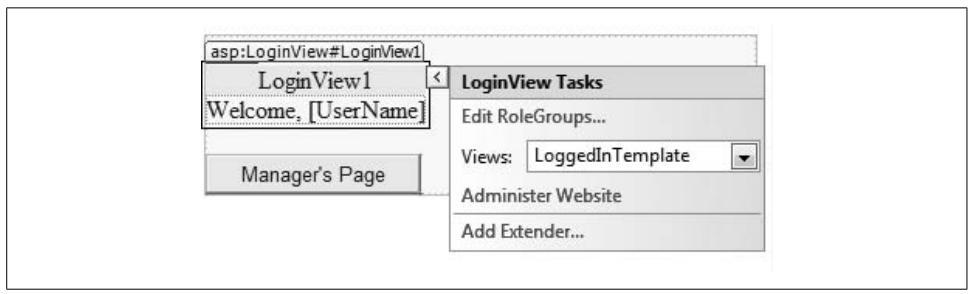


Figure 10-1. *Select the LoggedInTemplate view of the LoginView control. You're currently welcoming the user by using the LoginName, but you'll change that.*

Now, drag a `HyperLink` control from the Toolbox onto the `LoginView` control. Set its `Text` property to "Add Profile Info" and the `NavigateUrl` property to `ProfileInfo.aspx` (which you will create shortly). The Design view will look something like Figure 10-2.

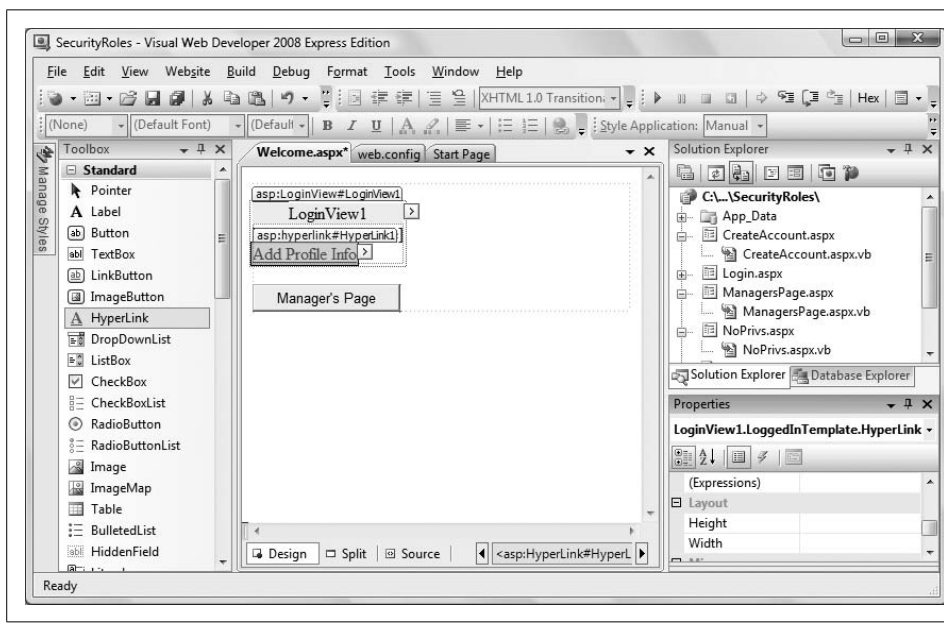


Figure 10-2. *Add the HyperLink control to the LoginView control to link to the page where you'll gather profile information.*

Create the page for gathering the profile information referred to in the `NavigateUrl` property of the `HyperLink`, *ProfileInfo.aspx*. Remember to check the box to place the code in a separate file. Add a table for layout to the page, and within the table, add labels and `TextBoxes`, as well as a Save button, as shown in Design view in Figure 10-3.

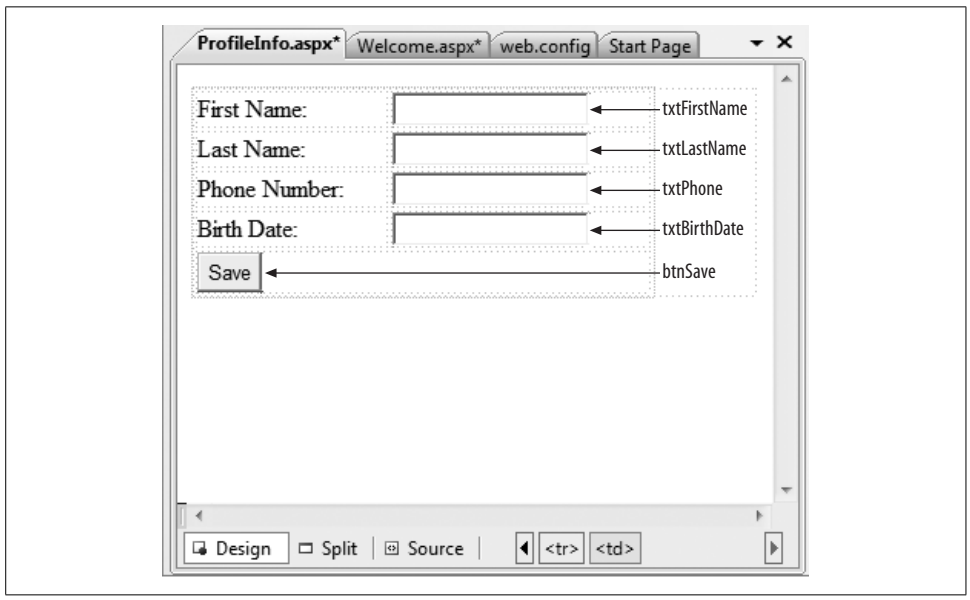


Figure 10-3. *The ProfileInfo page, shown here in Design view, is where users will enter their profile information.*

All that remains to be done is to add an event handler for the Save button. Double-click the Save button in Design view to open up a code skeleton for the `Click` event handler. Add the following highlighted code:

```
Protected Sub btnSave_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles btnSave.Click
    If Profile.IsAnonymous = False Then
        Profile.lastName = txtLastName.Text
        Profile.firstName = txtFirstName.Text
        Profile.phoneNumber = txtPhone.Text
        Profile.birthDate = CType(txtBirthDate.Text, System.DateTime)
    End If
    Response.Redirect("Welcome.aspx")
End Sub
```

Until the web site is built for the first time, the IDE will think all these Profile properties are invalid and underline them with the dreaded squiggly line. Either click the Build → Build Web Site menu item or just run the page.

The first line you added uses the `IsAnonymous` property of the `Profile` object. Anonymous users are those who haven't logged in yet. You can't set the profile properties if the user isn't logged in, so you need to check that first. The `Profile` object has properties that correspond to the properties you added in *web.config*.

To test that the `Profile` object has, in fact, stored this data, add a `Panel` control to the bottom of the Welcome page, as shown in Figure 10-4. Set the ID property of the `Panel` control to `pnlInfo` and set its `Visible` property to `False`, so that it will not normally display. Add an HTML table with three rows, each with a single cell containing a `Label` control. The `Labels` within the `Panel` control should be named `lblFullName`, `lblPhone`, and `lblBirthDate`. You should also set the `Text` properties of these Labels as shown in Figure 10-4.

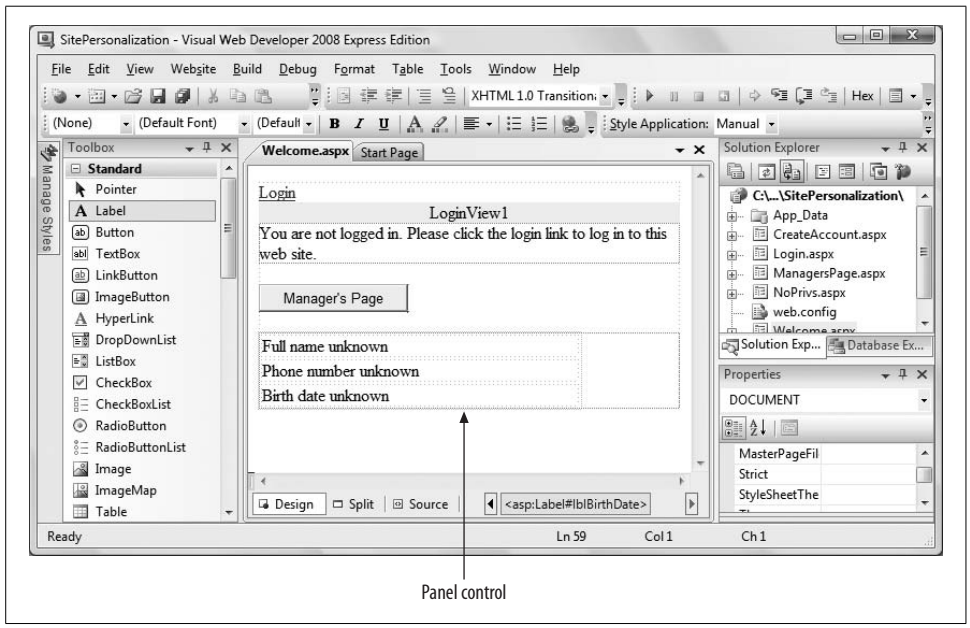


Figure 10-4. *After you've added the panel for displaying the profile information, the WelcomePage will look like this in Design view.*

Each `Label` control is initialized to say that the value is unknown (this is not normally needed, but we included it here to ensure that the data you see is retrieved from the `Profile` object). When the page is loaded, the event handler will check to see if you have `Profile` data for this user and, if so, assign that data to the appropriate controls and set the `Visible` property of `pnlInfo` to `True`.

To do this, you'll add a bit of code to the code-behind for the Welcome page, *Welcome.aspx.vb*, so that when the page loads, it will check to see if you have a profile, and if so, it will make the panel visible. You have previously created an event handler for the Page Load event for this page and added some code, so add the highlighted code from Example 10-2 to that existing event handler.

Example 10-2. *You'll use some simple code in the Page_Load handler for the Welcome page to retrieve and display the Profile information*

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    If User.Identity.IsAuthenticated = True Then
        btnManagersPage.Enabled = True
    Else
       btnManagersPage.Enabled = False
    End If

    If Not IsPostBack And _
            Profile.UserName IsNot Nothing And _
            Profile.IsAnonymous = False Then
        pnlInfo.Visible = True
        lblFullName.Text = Profile.firstName & " " & Profile.lastName
        lblPhone.Text = Profile.phoneNumber
        lblBirthDate.Text = Profile.birthDate.ToShortDateString( )
    Else
        pnlInfo.Visible = False
    End If
End Sub
```

When you start the application, you are asked to log in. Once logged in, a new hyperlink appears: Add Profile Info. This was created by the hyperlink you added to the `LoggedInTemplate` earlier. Clicking that link brings you to your new profile page, as shown in Figure 10-5.

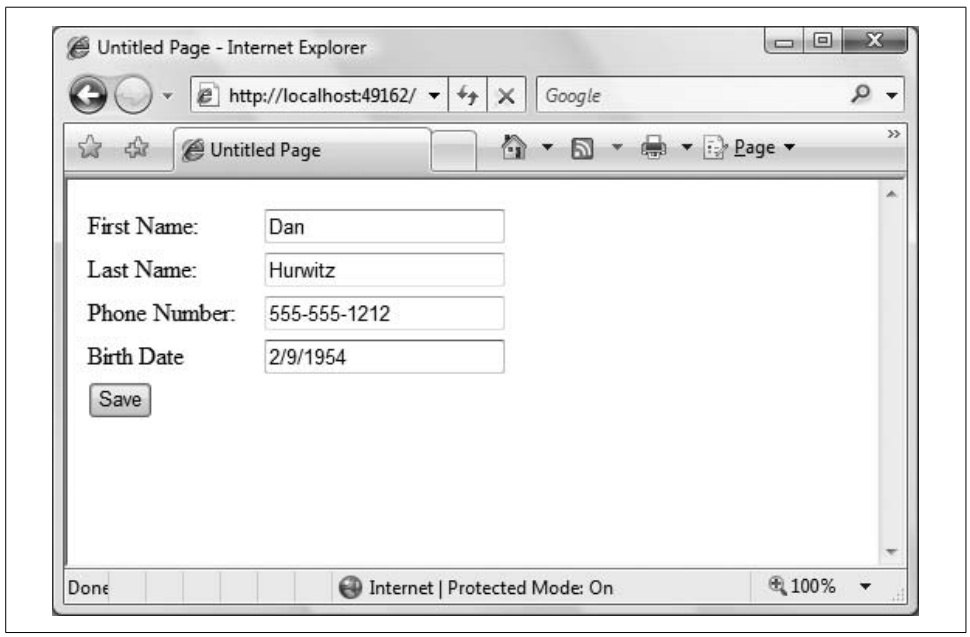


Figure 10-5. *The ProfileInfo page is a separate page with a simple form for gathering profile information.*

When you click Save and return to the Welcome page, the `Page_Load` event fires. The `Page_Load` contains a three-part `If` statement:

```
If Not IsPostBack And _
   Profile.UserName IsNot Nothing And _
   Profile.IsAnonymous = False Then
```

All parts of the `If` statement will evaluate to `True`: this page is not loading as a result of a postback (you're returning from the *ProfileInfo.aspx* page), the `UserName` value in the profile is not `Nothing` (you just set it), and the user is logged in, and thus not anonymous.

Your profile information is displayed, as shown in Figure 10-6.

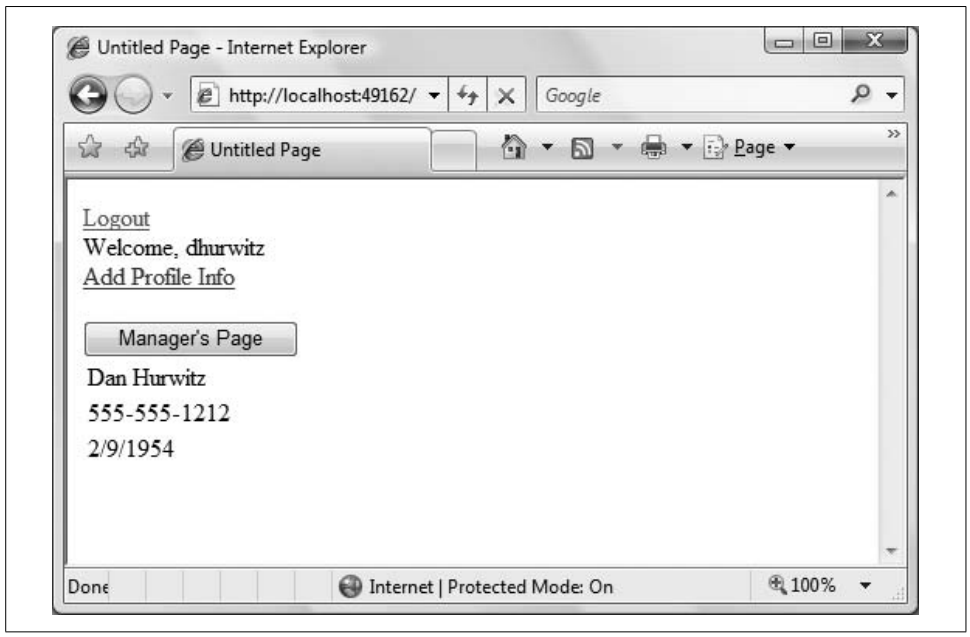


Figure 10-6. *After the user has entered and saved his profile information, it's displayed on the Welcome page.*

If you are logged in but have not yet entered any profile data, the default values will display, which are a blank string for string values and 1/1/0001 for the date.

Complex Data Types

The information we've saved so far in the profile has been simple, built-in data types, but of course you may want to save either user-defined types or collections. These are considered "complex data types" and require a bit of extra work, as you'll see with the *Sports* profile information in the very next example.

Close the browser and copy the previous example, SitePersonalization, to a new web site called *SitePersonalizationComplex*. Set the Welcome page as the Start page and run the site to verify that everything works.

In this web site, you will add a `CheckBoxList` control to the ProfileInfo page so users can select their favorite sports and store them in a profile using a `StringCollection` object.

Add a new property, called `Sports`, of type `StringCollection`, to the profile element in *web.config*, as indicated by the highlighted line of code in Example 10-3.

Example 10-3. *You can add a complex type to the profile element of web.config in the same place where you add basic types*

```
<profile enabled="true" defaultProvider="AspNetSqlProfileProvider">
    <properties>
        <add name="lastName" />
        <add name="firstName" />
        <add name="phoneNumber" />
        <add name="birthDate" type="System.DateTime" />
        <add name="Sports"
            type="System.Collections.Specialized.StringCollection" />
    </properties>
</profile>
```

Edit the page *ProfileInfo.aspx*. Add a row to the layout table above the Save button and put a `CheckBoxList` control in that row—name it `cblSports`. In Design view, click the Smart Tag of the `CheckBoxList` and click Edit Items…. Add several sports to the ListItem Collection Editor dialog box, as shown in Figure 10-7.

VB CHEAT SHEET

StringCollection Class

The `StringCollection` class, is a member of the `System.Collections.Specialized` namespace, is used to represent a collection of strings. Elements within the collection can be accessed using a zero-based integer index. A number of methods are provided for manipulating the collection, including the ability to add items, find the index of specific items, and remove items either by index or by specifying the string.

Now you need to enhance the event handler for the Save button to add the selected items to the new `Profile` property, as shown in the highlighted code in Example 10-4.

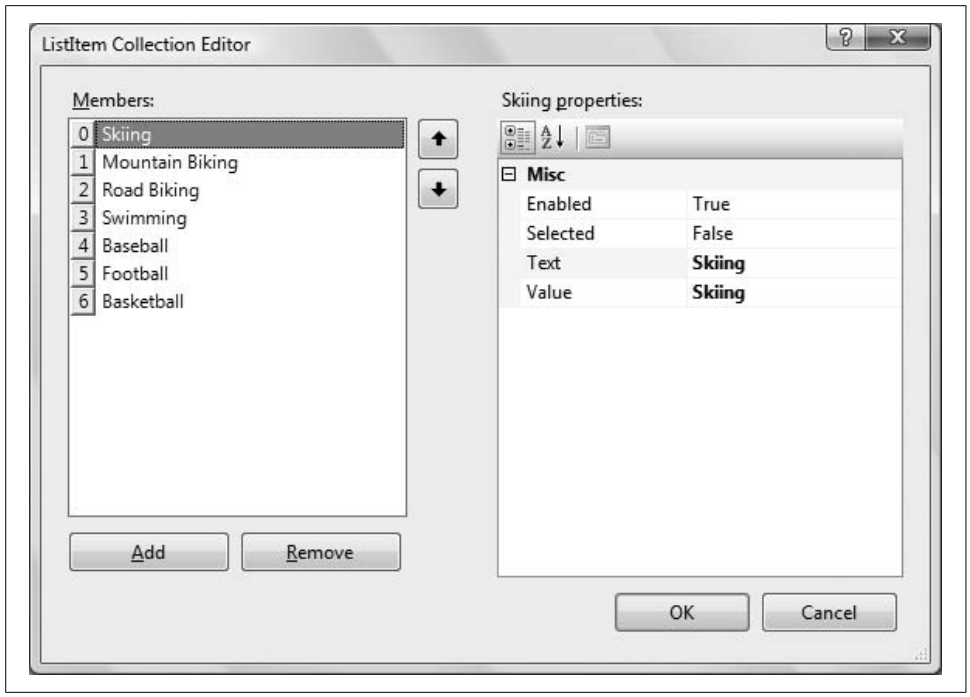


Figure 10-7. *Add these items to the CheckBoxList control.*

Example 10-4. *Modify the btnSave_Click event handler to process the complex Profile property*

```
Protected Sub btnSave_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles btnSave.Click
    If Profile.IsAnonymous = False Then
        Profile.lastName = txtLastName.Text
        Profile.firstName = txtFirstName.Text
        Profile.phoneNumber = txtPhone.Text
        Profile.birthDate = CType(txtBirthDate.Text, System.DateTime)

        Profile.Sports = New System.Collections.Specialized.StringCollection( )
        For Each item As ListItem In cblSports.Items
            If item.Selected Then
                Profile.Sports.Add(item.Value.ToString( ))
            End If
        Next

    End If
    Response.Redirect("Welcome.aspx")
End Sub
```

You also need to create an event handler for the Page_Load event, so the page will open with the user's up-to-date profile information. Create that event the same way you did earlier, and then add the body of the method, highlighted in Example 10-5.

Example 10-5. *The Page_Load handler for ProfileInfo.aspx.vb guarantees that the user will see his current profile information when the page loads*

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    If Not IsPostBack And Profile.UserName IsNot Nothing Then
        If Profile.IsAnonymous = False Then
            txtLastName.Text = Profile.lastName
            txtFirstName.Text = Profile.firstName
            txtPhone.Text = Profile.phoneNumber
            txtBirthDate.Text = Profile.birthDate.ToShortDateString( )
        End If

        If Profile.Sports IsNot Nothing Then
            For Each item As ListItem In cblSports.Items
                For Each profileString As String In Profile.Sports
                    If item.Text = profileString Then
                        item.Selected = True
                    End If
                Next
            Next
        End If
    End If
End Sub
```

Each time you navigate to the Profile page, the values are updated from the existing profile (if any) in Page_Load, and you are free to change them and save the new values, as shown in Figure 10-8.

However, after you save this page and go back to the Welcome page, the sports selections are not displayed. To do so, add a ListBox control to the Welcome page, called lbSports, to the already existing Panel control pnlInfo, as shown in Figure 10-9. The selections will be displayed in lbSports.

Modify the existing Page_Load handler in *Welcome.aspx.vb* to bind the contents of the Profile.Sports property to the ListBox by adding the highlighted code from Example 10-6.

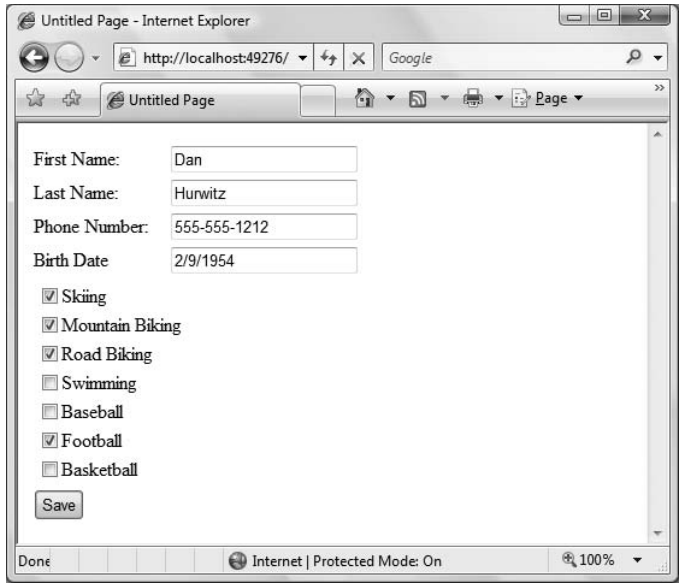


Figure 10-8. *The ProfileInfo page now has a CheckBoxList control showing complex profile properties, and furthermore, the Profile object retains the data, so that it can be reloaded whenever the user returns to this page.*

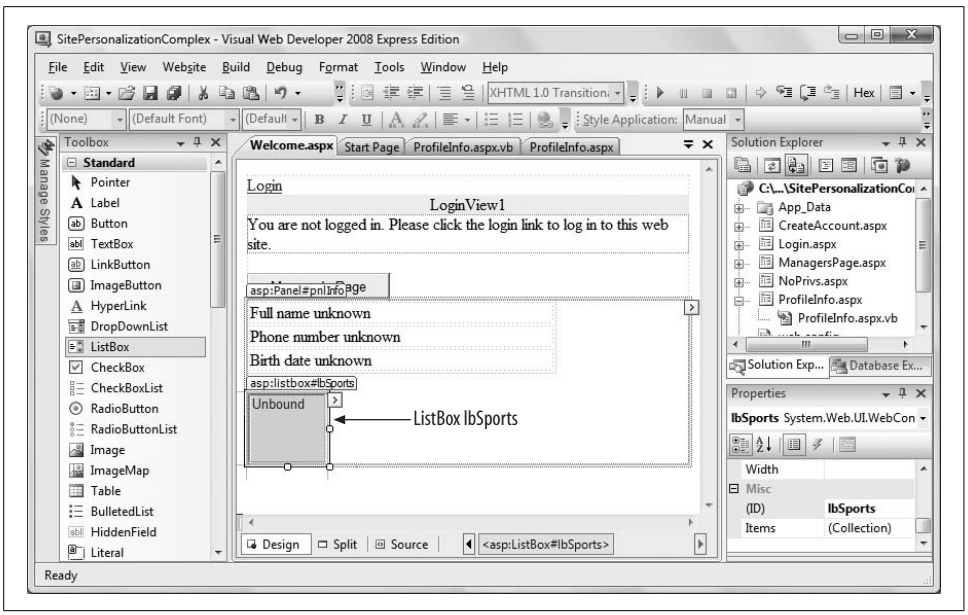


Figure 10-9. *Add a ListBox control to the Welcome page to display the complex profile property.*

Example 10-6. *Modify the Page_Load handler for Welcome.aspx.vb to bind the Profile.Sports data to the ListBox*

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    If User.Identity.IsAuthenticated = True Then
        btnManagersPage.Enabled = True
    Else
        btnManagersPage.Enabled = False
    End If

    If Not IsPostBack And _
            Profile.UserName IsNot Nothing And _
            Profile.IsAnonymous = False Then
        pnlInfo.Visible = True
        lblFullName.Text = Profile.firstName & " " & Profile.lastName
        lblPhone.Text = Profile.phoneNumber
        lblBirthDate.Text = Profile.birthDate.ToShortDateString( )

        If Profile.Sports IsNot Nothing Then
            For Each sport As String In Profile.Sports
                lbSports.Items.Add(sport)
            Next
        End If
    Else
        pnlInfo.Visible = False
    End If

End Sub
```

When you click Save in the Profile page and return to the Welcome page, your saved profile information is displayed, as shown in Figure 10-10.

Anonymous Personalization

Often you will want to allow users to use your site for a while before logging in. Along the way, the user may fill in information that you will want to store in the user's profile once the user is logged in. Imagine, for example, that the user has a shopping cart. Your use model may be that anonymous users may add items to the cart. If they want the cart to persist after they leave, or if they want to buy the items, they must log in; otherwise, you'll toss the cart after their session times out.

You need a way to store her anonymous profile, and, more important, you need a way to merge that anonymous profile with her actual profile once you know who she really are. No problem; ASP.NET provides for that very circumstance.

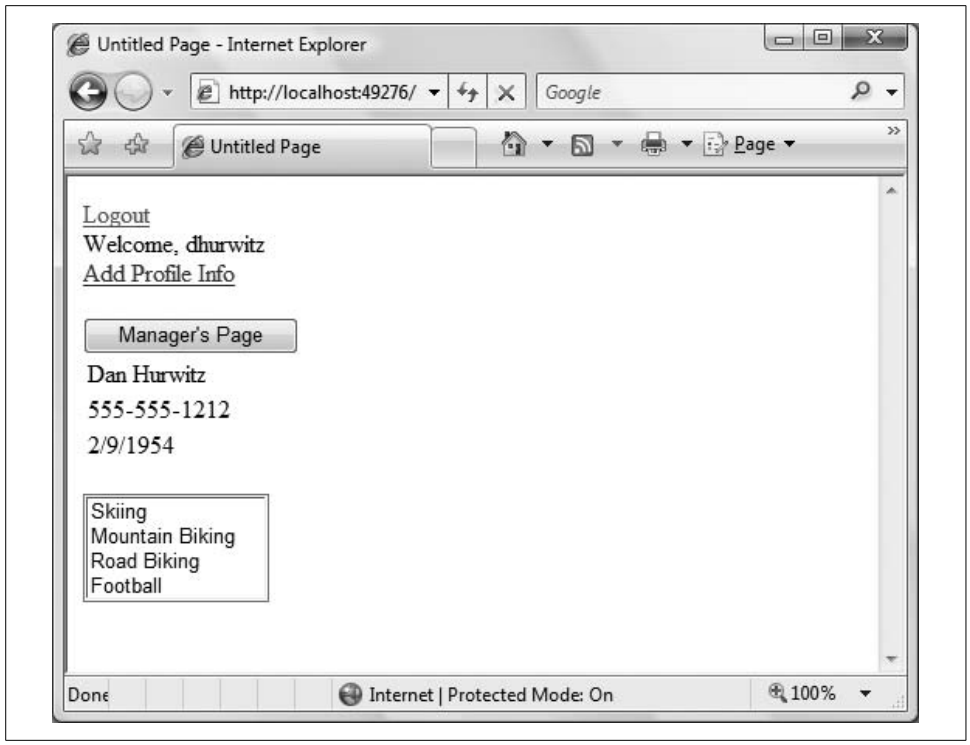


Figure 10-10. *The Welcome page now displays the contents of Profile.Sports for logged-in users.*

Adding an Anonymous Profile

Copy the previous example, SitePersonalizationComplex, to a new web site called AnonymousPersonalization. Set *Welcome.aspx* as the Start page and run the site to verify that everything is working.

To enable anonymous personalization, add the highlighted lines from Example 10-7 to your *web.config* file. Also highlighted in Example 10-7 is a line adding the `allowAnonymous` attribute to the `Sports` profile property.

Example 10-7. *You need to make a couple of small edits to web.config to enable anonymous personalization*

```
<?xml version="1.0"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
    <appSettings/>
    <connectionStrings/>
    <system.web>
        <anonymousIdentification enabled="true" />
        <roleManager enabled="true" />
        <compilation debug="true" strict="false" explicit="true"/>
        <profile enabled="true" defaultProvider="AspNetSqlProfileProvider">
```

Example 10-7. *You need to make a couple of small edits to web.config to enable anonymous personalization (continued)*

```
            <properties>
                <add name="lastName" />
                <add name="firstName" />
                <add name="phoneNumber" />
                <add name="birthDate" type="System.DateTime" />
                <add name="Sports"
                    type="System.Collections.Specialized.StringCollection"
                    allowAnonymous="true" />
            </properties>
        </profile>
        <pages>
           <!--stuff omitted for brevity --!>
        </pages>
        <authentication mode="Forms"/>
    </system.web>
</configuration>
```

Because anonymous users can now save and see the `Sports` profile property, but *only* the `Sports` profile property, you need to modify the Welcome and ProfileInfo pages.

Redesign your *Welcome.aspx* page in two ways: first, move the `HyperLink` that leads to the Profile Information page outside of the `LoginView` control. Second, move the ListBox (`lbSports`) outside the Panel `pnlInfo`. Thus, you can see both of these features whether you are logged in or not. Also, change the text on the Add Profile Info HyperLink to just "Profile Info", as you will be using this link to add and edit the profile info.

You must modify the `Page_Load` event handler from *Welcome.aspx.vb*, shown previously in Example 10-6, to properly display the anonymous information. The modified `Page_Load` handler is shown in Example 10-8.

Example 10-8. *Modify the Page_Load handler in Welcome.aspx.vb to properly display the anonymous profile properties*

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    If User.Identity.IsAuthenticated = True Then
        btnManagersPage.Enabled = True
    Else
        btnManagersPage.Enabled = False
    End If

    If Not IsPostBack And _
            Profile.UserName IsNot Nothing Then
        If Profile.IsAnonymous = False Then
            pnlInfo.Visible = True
            lblFullName.Text = Profile.firstName & " " & Profile.lastName
            lblPhone.Text = Profile.phoneNumber
            lblBirthDate.Text = Profile.birthDate.ToShortDateString( )
        End If
```

Example 10-8. *Modify the Page_Load handler in Welcome.aspx.vb to properly display the anonymous profile properties (continued)*

```
        If Profile.Sports IsNot Nothing Then
           For Each sport As String In Profile.Sports
               lbSports.Items.Add(sport)
           Next
        End If
    Else
        pnlInfo.Visible = False
    End If
End Sub
```

When an anonymous user fills in the profile information, the user will automatically be assigned a Globally Unique Identifier (GUID), and an entry will be made in the database for that ID. However, note that only those properties marked with `allowAnonymous` can be stored, so you must modify your `btnSave_Click` event handler in *ProfileInfo.aspx.vb*. Bracket the entries for all the profile elements *except* `Sports` in an `If` statement that tests whether the user is currently `Anonymous`. The new `btnSave_Click` event handler for *ProfileInfo.aspx.vb* is shown in Example 10-9.

Example 10-9. *The Save event handler for ProfileInfo.aspx.vb with the code for saving Sports property moved outside of the test for IsAnonymous*

```
Protected Sub btnSave_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles btnSave.Click
    If Profile.IsAnonymous = False Then
        Profile.lastName = txtLastName.Text
        Profile.firstName = txtFirstName.Text
        Profile.phoneNumber = txtPhone.Text
        Profile.birthDate = CType(txtBirthDate.Text, System.DateTime)
    End If

    Profile.Sports = New System.Collections.Specialized.StringCollection( )
    For Each item As ListItem In cblSports.Items
        If item.Selected Then
            Profile.Sports.Add(item.Value.ToString( ))
        End If
    Next

    Response.Redirect("Welcome.aspx")

End Sub
```

The effect of the new code shown in Example 10-9 is that you check whether the `IsAnonymous` property is `False`. If it is, then you are dealing with a logged-in user, and you can get all of the properties; otherwise, you can get only those that are allowed for anonymous users.

Modify the *ProfileInfo* page so that the nonanonymous data is in a panel that will be invisible for users who are not logged in. The simplest way to do this may be to switch to Source view and bracket the nonanonymous code inside a panel (don't

forget to end the anonymous table before ending the panel, and then insert another opening table tag), as shown in Example 10-10.

Example 10-10. *Add a panel to ProfileInfo.aspx for hiding the nonanonymous information*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="ProfileInfo.aspx.vb"
    Inherits="ProfileInfo" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Panel ID="pnlNonAnonymousInfo" runat="server" >
            <table>
                <tr>
                    <td>First Name:</td>
                    <td>
                        <asp:TextBox ID="txtFirstName" runat="server" />
                    </td>
                </tr>
                <tr>
                    <td>Last Name:</td>
                    <td>
                        <asp:TextBox ID="txtLastName" runat="server" />
                    </td>
                </tr>
                <tr>
                    <td>Phone Number:</td>
                    <td>
                        <asp:TextBox ID="txtPhone" runat="server" />
                    </td>
                </tr>
                <tr>
                    <td>Birth Date:</td>
                    <td>
                        <asp:TextBox ID="txtBirthDate" runat="server" />
                    </td>
                </tr>
            </table>
        </asp:Panel>
        <table>
            <tr>
                <td colspan="2">
                    <asp:CheckBoxList ID="cblSports" runat="server" >
                        <asp:ListItem>Skiing</asp:ListItem>
                        <asp:ListItem>Mountain Biking</asp:ListItem>
                        <asp:ListItem>Road Biking</asp:ListItem>
```

Example 10-10. *Add a panel to ProfileInfo.aspx for hiding the nonanonymous information (continued)*

```
                        <asp:ListItem>Swimming</asp:ListItem>
                        <asp:ListItem>Baseball</asp:ListItem>
                        <asp:ListItem>Football</asp:ListItem>
                        <asp:ListItem>Basketball</asp:ListItem>
                    </asp:CheckBoxList>
                </td>
            </tr>
            <tr>
                <td colspan="2">
                    <asp:Button ID="btnSave" runat="server" Text="Save"/>
                </td>
            </tr>
        </table>
    </div>
    </form>
</body>
</html>
```

In order to hide this panel if the user is anonymous, edit the `Page_Load` event handler in *ProfileInfo.aspx.vb*, as shown in Example 10-11. This code controls the visibility of `pnlNonAnonymousInfo` based on whether or not the user is anonymous. Under any circumstances, the Sports preferences will be displayed.

Example 10-11. *Modify the Page_Load handler in ProfileInfo.aspx.vb to properly display anonymous profile properties*

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    If Not IsPostBack And Profile.UserName IsNot Nothing Then
        If Profile.IsAnonymous = True Then
            pnlNonAnonymousInfo.Visible = False
        Else
            pnlNonAnonymousInfo.Visible = True
            txtLastName.Text = Profile.lastName
            txtFirstName.Text = Profile.firstName
            txtPhone.Text = Profile.phoneNumber
            txtBirthDate.Text = Profile.birthDate.ToShortDateString( )
        End If

        If Profile.Sports IsNot Nothing Then
            For Each item As ListItem In cblSports.Items
                For Each profileString As String In Profile.Sports
                    If item.Text = profileString Then
                        item.Selected = True
                    End If
                Next
            Next
        End If ' close for If Profile.Sports IsNot Nothing
    End If ' close for If Not IsPostBack And Profile.UserName IsNot Nothing
End Sub
```

Run the application. Do *not* log in, but do click the Profile Info link. Select a few sports and click Save. When you return to the Welcome page, you are still not logged in, but your selected sports are displayed, as shown in Figure 10-11.

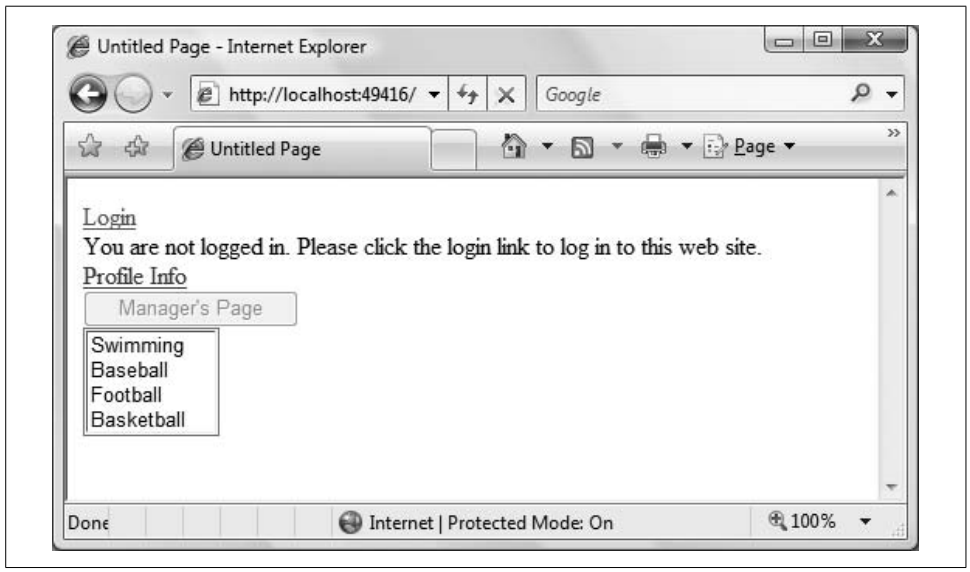


Figure 10-11. *The Profile information for an anonymous user is reflected on the Welcome page, without logging in.*

Migrating Anonymous Data to an Actual User's Record

When the user *does* log in, you must migrate the Profile data you've accumulated for the anonymous user to the authenticated user's record (so that, for example, shopping cart items are not lost). You do this by writing a global handler in *global.asax*. The *global.asax* file contains event handlers that apply to the web site as a whole. So far you've worked only with events that happen in a page, like a button click or page load. The *global.asax* file lets you write handlers for events like when a request is first received, when a session starts, when a user is authenticated (logged in), and so on.

Your project probably does not yet have a *global.asax* file, so click the Website → Add New Item menu item. One of your choices will be Global Application Class, and it will default to the name *global.asax*. Click Add to accept the default name.

When a user logs in to an ASP.NET application, the `MigrateAnonymous` event is fired automatically. You can handle that event with a method in *global.asax*. The code to do so is listed in Example 10-12.

Example 10-12. *The MigrateAnonymous event handler in global.asax applies to all the pages in your application*

```
Sub Profile_MigrateAnonymous(ByVal sender As Object, _
  ByVal e As ProfileMigrateEventArgs)
    Dim anonymousProfile As ProfileCommon = Profile.GetProfile(e.AnonymousID)
    If anonymousProfile IsNot Nothing And _
     anonymousProfile.Sports IsNot Nothing Then
        For Each str As String In anonymousProfile.Sports
            Profile.Sports.Remove(str) ' remove duplicates
            Profile.Sports.Add(str)
        Next
    End If
End Sub
```

The event argument for `MigrateAnonymous` is of type `ProfileMigrateEventArgs`. This event argument has a property called `AnonymousID`, which contains the ID of the anonymous user (this is all done automatically). The first step, then, in this method is to get the profile that corresponds to the `AnonymousID` of the anonymous user:

```
Dim anonymousProfile As ProfileCommon = Profile.GetProfile(e.AnonymousId)
```

If there is such a profile—that is, if the profile is not `Nothing`—then you know that there is a matching anonymous profile, and that you may choose whatever data you need from that profile. In this case, you copy over the `Sports` collection by using the `For Each` loop to iterate through the `Sports` collection. Each `Sport` is added to the current profile for the logged-in user, after first removing it to avoid duplicates.

Themes and Skins

Many users like to personalize their favorite web sites by setting the look and feel to meet their own aesthetic preferences. ASP.NET supports that requirement with "themes."

A *theme* is a collection of skins. A *skin* describes how a control should look. A skin can define style sheet attributes, images, colors, and so forth.

Having multiple themes allows your users to choose how they want your site to look by switching from one set of skins to another at the click of a button. Combined with personalization, your site can remember the look and feel each user prefers.

There are two types of themes. The first, called *stylesheet themes*, define styles that can be overridden by the page or control. These are, essentially, equivalent to CSS style sheets. The second type, called *customization themes*, cannot be overridden. You set a stylesheet theme by adding the `StyleSheetTheme` attribute to the `Page` directive, and, similarly, you set a customization theme by setting the `Theme` attribute in the `Page` directive.

In any given page, the properties for the controls are set in this order:

- Properties are applied first from a stylesheet theme.
- Properties are then overridden based on properties set in the control.
- Properties are then overridden based on a customization theme.

Thus, the customization theme is guaranteed to have the final word in determining the look and feel of the control. Overriding a property replaces the current version of the property with a new version.

Skins themselves come in two flavors: *default skins* and *explicitly named skins*. Thus, you might create a Labels skin file called *Labels.skin* with this declaration:

```
<asp:Label runat="server"
    ForeColor="Blue" Font-Size="Large"
    Font-Bold="True" Font-Italic="True" />
```

This is a default skin for all `Label` controls. It looks just like the declaration of an ASP.NET `Label` control (minus the ID attribute), but it is housed in a *.skin* file and, thus, is used to define the look and feel of all `Label` objects within that skin file's theme.

In addition, however, you might decide that some labels must be red. To accomplish this, create a second skin, but assign this skin a `SkinID` property:

```
<asp:Label runat="server" SkinID="RedLabel"
ForeColor="Red" Font-Size="Large"
Font-Bold="True" Font-Italic="True" />
```

Any label that does not have a `SkinID` attribute will get the default skin; any label that sets `SkinID="RedLabel"` will get your named skin.

The steps to providing a personalized web site are as follows:

1. Create the test site.
2. Organize your themes and skins.
3. Enable themes and skins for your site.
4. Specify themes declaratively if you wish.

Create the Test Site

To demonstrate the use of themes and skins, copy the previous example web site, AnonymousPersonalization, to a new web site called *Themes*. Set the start page to *Welcome.aspx* and test the application to make sure it still works as expected.

The first thing to do is to add some controls whose look and feel you can set.

Open *Welcome.aspx*, create a table for layout with two rows and four columns underneath `lbSports`, and drag on some new controls, as shown in Figure 10-12.

There are four labels: `lblListBox`, `lblRadioButtonList`, `lblCalendar`, and `lblTextBox`. Each of these labels provides a caption for the neighboring control, a `ListBox`, a `RadioButtonList`, a `Calendar`, and a `TextBox`, respectively. Use the default properties for these four controls, other than the IDs and Text properties of the Label controls.

You'll also need to click the Smart Tag for both `ListBox1` and `RadioButtonList1`. For each of these, choose Edit Items. In the ListItem Collection Editor, add four items to `ListBox1` and six items to `RadioButtonList1`, the result of which is shown in Figure 10-12. In this example, the ListItems are named First Item, RadioButton1, and so on. These are not the default names; they are just chosen to make the example clear.

You will use themes to change the look and feel of the new controls.

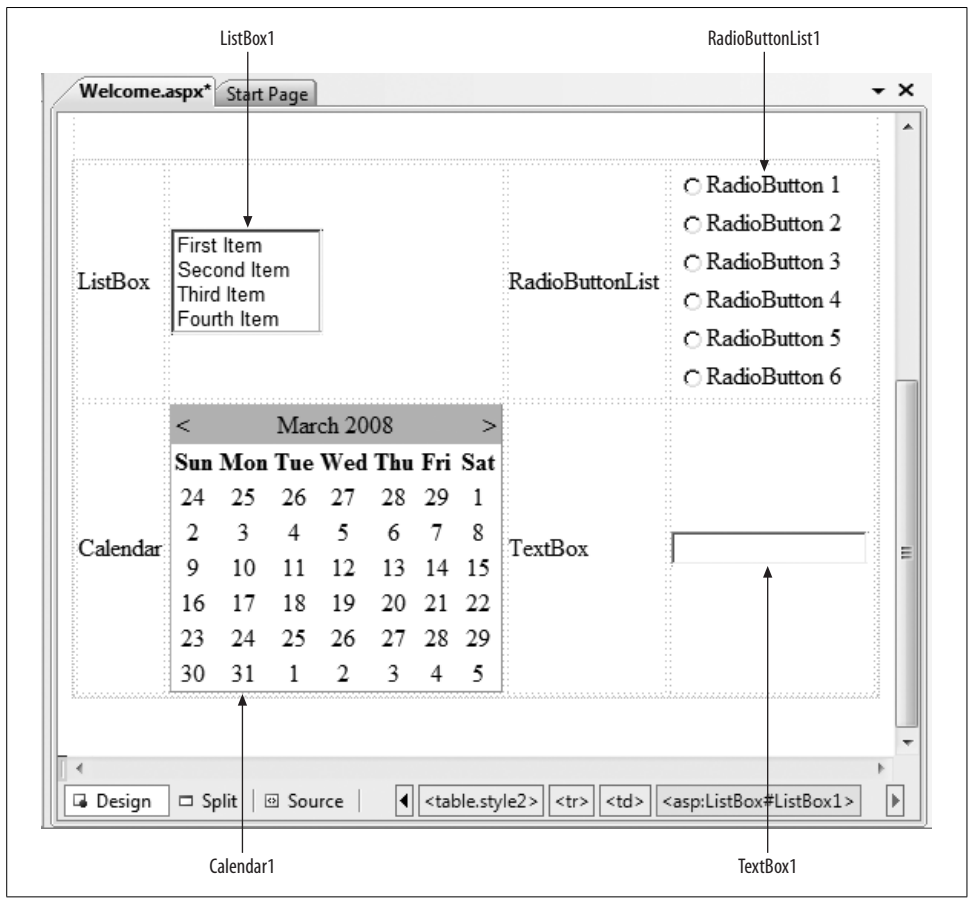


Figure 10-12. *The Themes test page isn't very functional, but it has a variety of controls so you can see the effects of the themes.*

Organize Site Themes and Skins

Themes are stored in your project in a special folder named `App_Themes`. To create this folder, go to Solution Explorer, right-click the web site, and choose Add ASP.NET Folder → Theme, as shown in Figure 10-13. Name the new folder *Dark Blue*—the folder *App_Themes* will be created automatically, with a Theme folder named *Dark Blue* immediately under it. Right-click App_Themes in the Solution Explorer and again select Add ASP.NET Folder → Theme. Create a second theme folder, named *Psychedelic*.

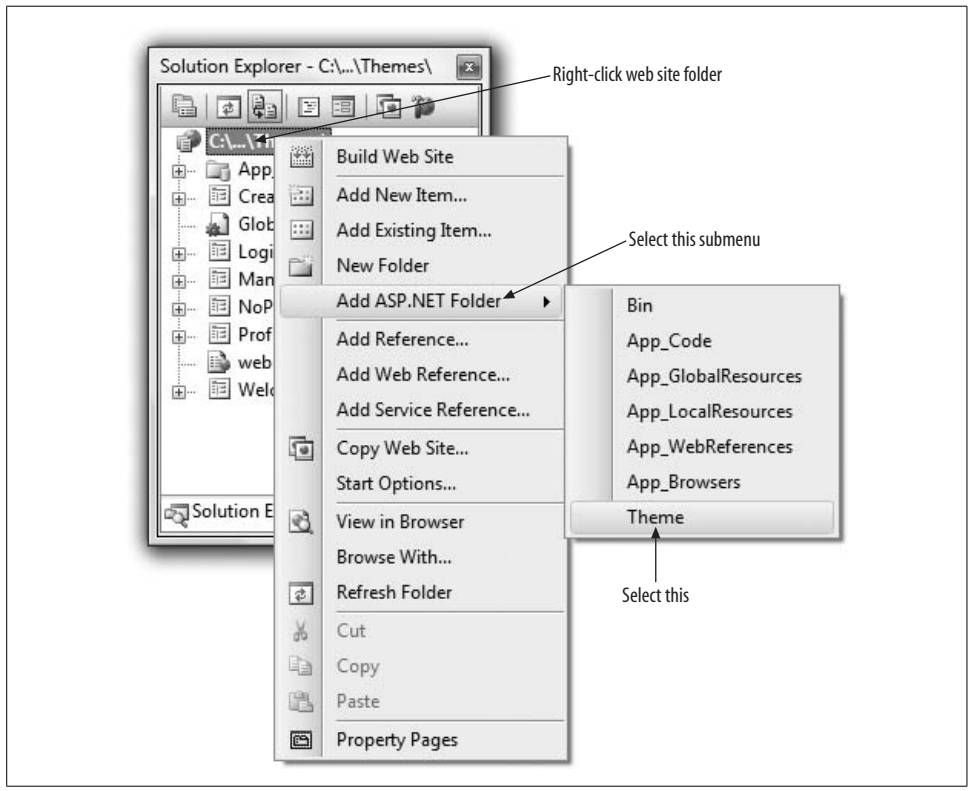


Figure 10-13. *Adding a Theme folder to a web site is just a right-click away.*

Right-click the *Dark Blue* theme folder and choose Add New Item. From the template lists, choose Skin File and name it *Button.skin* (to hold all the button skins for your Dark Blue theme), as shown in Figure 10-14.

Each skin file is just a text file that contains a definition for the control type, but with no ID. Thus, your *Button.skin* file for the Dark Blue theme might look like this:

```
<asp:Button runat="server"
    ForeColor="Blue"
    Font-Size="Large" Font-Bold="True" Font-Italic="True" />
```

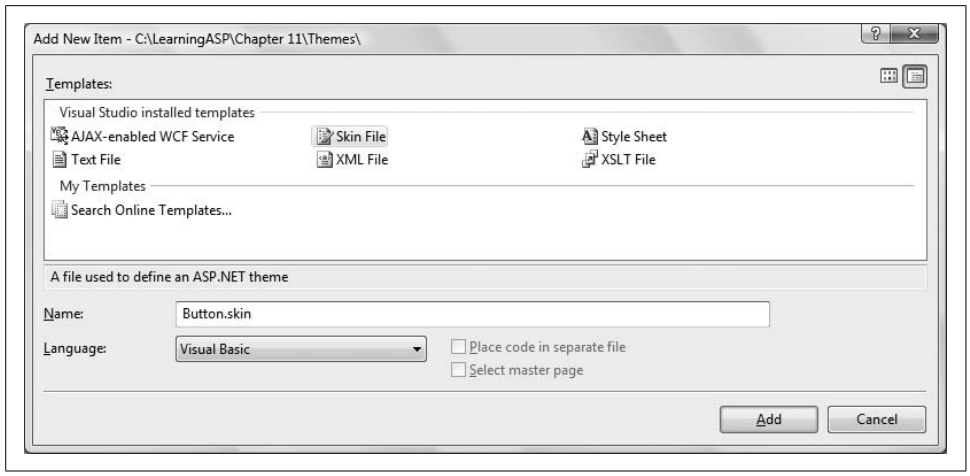


Figure 10-14. *You add a new Skin file with the Add New Item dialog.*

Create skin files for each of the following types in both themes:

- *Button.skin*
- *Calendar.skin*
- *Label.skin*
- *ListBox.skin*
- *RadioButtonList.skin*
- *TextBox.skin*

The name of the skin file doesn't have to correspond to the type of control referenced in the file, but it does simplify site maintenance. In fact, you don't even need to put each control in a separate skin file, so you could put them all into a single skin file.

At this point, your Solution Explorer should look more or less like Figure 10-15.

The rest of these skin files are empty right now, so you should create some skins for the purpose of this example. Copy the code from *Button.skin*, and paste it into *Calendar.skin*. Change `asp:Button` to `asp:Calendar`, and now the calendar skin file is ready to go. Use the same procedure for all the skin files in the Dark Blue theme. Then, do the same for the skin files in the Psychedelic theme, but feel free to change the color and size of the fonts in these files as you see fit.

Enable Themes and Skins

To let your users choose the theme they like and have their preference stored in their profile, you need only to add a single line to the `properties` subelement in the `profile` element of *web.config*:

```
<add name="Theme" />
```

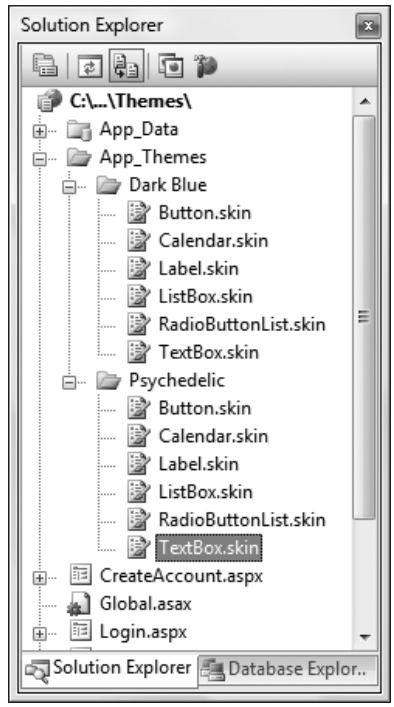


Figure 10-15. *After you've created all of the skin files, you can see them in Solution Explorer.*

Strictly speaking, adding this line to *web.config* is not necessary to enable themes on a site. It is only necessary to enable *saving* themes to a user profile.

Save and rebuild your application.

In order to run an ASP.NET web site, the application must be *built*, or compiled. When you run a web site from within the IDE, it is built automatically, as indicated on the status line at the bottom of the window. It is also possible to build the application without running it by clicking the Build menu item and choosing one of the options: Build Page, Build Web Site, or Rebuild Web Site. Normally, a page or code file is not rebuilt if nothing has changed. The Rebuild menu item forces a rebuild of all components.

Sometimes it is helpful to build the site without running it just to check for syntax and compiler errors. Also, sometimes the IDE gets confused until the app is rebuilt, at which point everything is known to the system.

Specify Themes for Your Page

You can set the themes on your page either declaratively or programmatically. For example, to set a customization theme declaratively for *Welcome.aspx*, add the `Theme` attribute to the `Page` directive:

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Welcome.aspx.vb"
    Inherits="Welcome" Theme="Dark Blue" %>
```

Set *Welcome.aspx* as the start page. Run the app now and you will see the Dark Blue theme applied, as shown in Figure 10-16. (Obviously the monochromatic printed book will not show the colors in full splendor.)

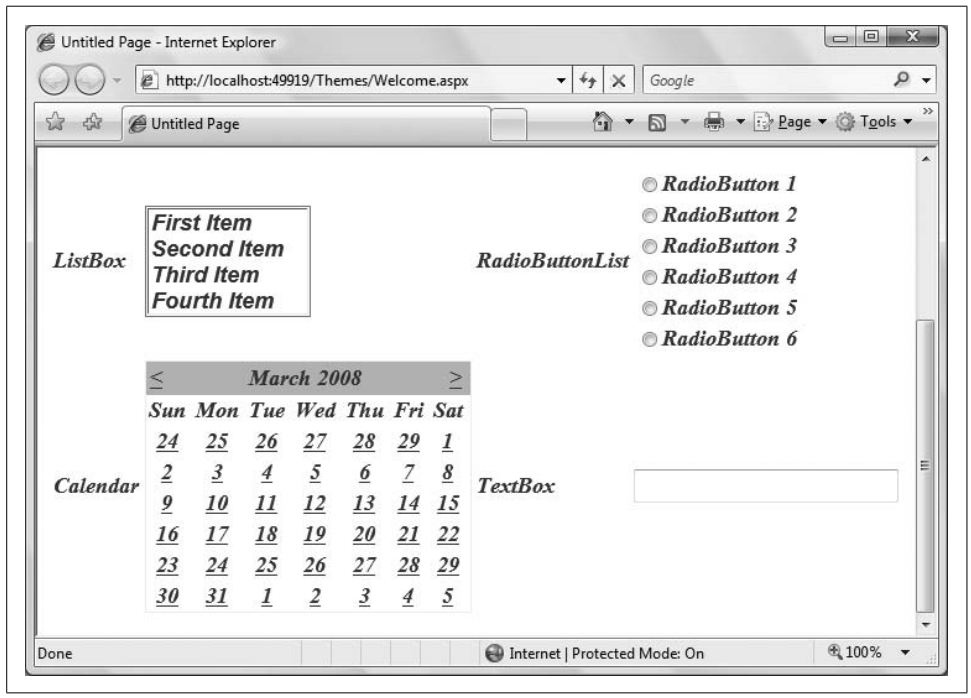


Figure 10-16. *The page is rendered using the Dark Blue theme as a result of adding the Theme attribute to the Page directive.*

You can also set the theme programmatically, either by coding it directly onto the page or (even better) by setting it from the user's profile.

StyleSheet themes, as opposed to customization themes, are set by overriding the `StyleSheetTheme` property for the page. IntelliSense will help you with this. Open *Welcome.aspx.vb* and scroll to the bottom of the class. Type the word **`overrides`** just above the `End Class` statement and all the overridable members are shown.

Start typing *sty* and IntelliSense will scroll to the property you want: `StyleSheetTheme`, as shown in Figure 10-17.

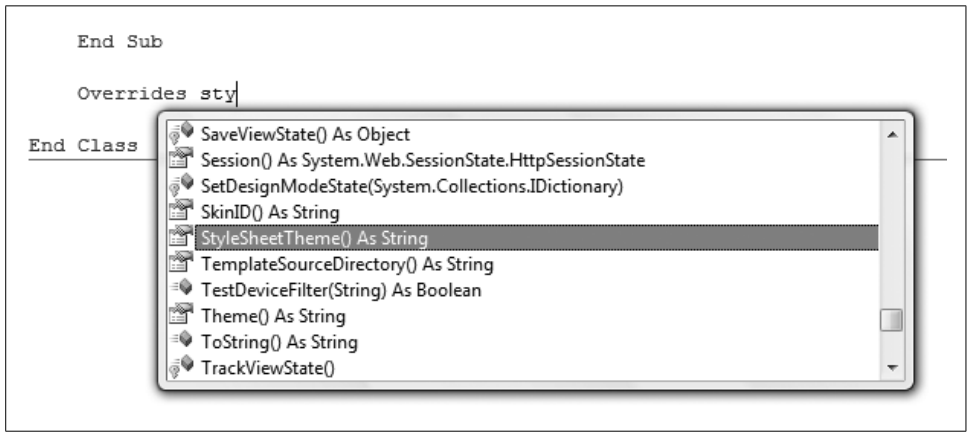


Figure 10-17. *IntelliSense helps you to override the StyleSheetTheme property.*

Once IntelliSense finds the property you want, press the Tab key to accept it. It will create a code skeleton for a property called `StyleSheetTheme`.

VB CHEAT SHEET

Properties

Properties in .NET are implemented using special syntax. This is true for all the properties you've used so far, but you haven't had to concern yourself with how a property is actually defined behind the scenes. Like all properties in .NET, `StyleSheetTheme` is implemented with two special methods: one, called `Get`, for retrieving the value of the property; another, called `Set`, for setting the value of the property. These special methods are referred to variously as the *accessors* or the *getter* and *setter*.

When your code retrieves the value of a property, as in the following:

```
Dim i as Integer = SomeProperty
```

The getter for that property is implicitly called. You don't have to actually call it. Similarly, if you set the value of a property, the setter is implicitly called.

If a property needs to be read-only, then omit the setter. If for some reason you want to disallow using the value of the property, you could omit the getter.

This code skeleton includes default code for both the getter and setter. Replace that default code with the highlighted code in Example 10-13.

Example 10-13. *Setting the StyleSheetTheme property*

```
Public Overrides Property StyleSheetTheme( ) As String
    Get
        If Profile.IsAnonymous = False _
        And Profile.Theme IsNot Nothing Then
            Return Profile.Theme
        Else
            Return "Dark Blue"
        End If
    End Get
    Set(ByVal value As String)
        Profile.Theme = value
    End Set
End Property
```

If you are going to set a *customization* theme programmatically, however, you must do so from the page's `PreInit` event handler, because the theme must be set before the controls are created. Chapter 7 covered when the various page events occur. A `Page_PreInit` event handler is created the same way a `Page_Load` event handler is created: select (Page Events) in the left drop-down and PreInit in the right. Create the `PreInit` event handler and type in the bolded code from Example 10-14.

Example 10-14. *The PreInit event handler in Welcome.aspx.vb sets the customization theme before the page loads*

```
Protected Sub Page_PreInit(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.PreInit
    If Profile.IsAnonymous = False Then
        Page.Theme = Profile.Theme
    End If
End Sub
```

Setting the theme in `PreInit` creates a bit of a difficulty when you want to allow the user to change the theme at runtime. If you add a control that lets the user select a theme, when that page posts back with a new theme, the `PreInit` code runs *before* the event handler for the button that changes the theme (see the discussion of the page life cycle in Chapter 7). So, by the time the code runs to change the theme, the controls have already been drawn.

To overcome this, you must, unfortunately, refresh the page again. This can be done easily enough by calling the `Server.Transfer` method to transfer to itself from the event handler, which also sets the theme.

To see this, add two buttons to *Welcome.aspx*, labeled Psychedelic and Dark Blue, with ID's of `btnPsychedelic` and `btnDarkBlue`, respectively. You will want both buttons to share the same event handler, `Set_Theme`, shown in Example 10-15. An easy way to have the IDE set up that event handler for you is to switch to Design view and single click one of the buttons to select it. Click the lightning bolt in the Properties window to go to the events, click next to the `Click` event, and type in the method

name Set_Theme. You are now ready to type in the event handler. Once the method exists in your code-behind, you can hook the other button to the method by clicking the button and selecting the method from the drop-down next to the Click event in the Properties window.

In the handler, you'll cast the sender to the Button type (using CType) and check its Text property, setting the theme appropriately.

Example 10-15. *The Set_Theme method Button Click event handler in Welcome.aspx.vb handles the buttons that set the theme*

```
Protected Sub Set_Theme(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles btnPsychedelic.Click, btnDarkBlue.Click
    Dim btn As Button = CType(sender, Button)
    If btn.Text = "Psychedelic" Then
        Profile.Theme = "Psychedelic"
    Else
        Profile.Theme = "Dark Blue"
    End If

    Server.Transfer(Request.FilePath)
End Sub
```

What's going on here is that you're creating a variable called btn that is a reference to the button that raised the event (sender). If the Text property of the sender is "Psychedelic," you set the Psychedelic theme. If not, the event must have come from the Dark Blue button, and that theme is set instead.

There is one more problem. If you run this page and try to set the theme before you are logged in, an exception will result. The Theme property cannot be set for an anonymous user.

To prevent this from happening, you will hide the two theme buttons unless the user is logged in. To do so, add the following two lines to the Page_Load method of *Welcome.aspx.vb*:

```
btnDarkBlue.Visible = Not Profile.IsAnonymous
btnPsychedelic.Visible = Not Profile.IsAnonymous
```

These lines set the Visible property of the Buttons to the opposite of the IsAnonymous property. If the user is logged in, IsAnonymous is False, so the button's Visible property is set True. Now when the user logs in, the page will look something like that shown in Figure 10-18.

When the user is not logged in, the Welcome page's default theme will be used. Once the user sets a theme, that theme will be used when you return to the Welcome page.

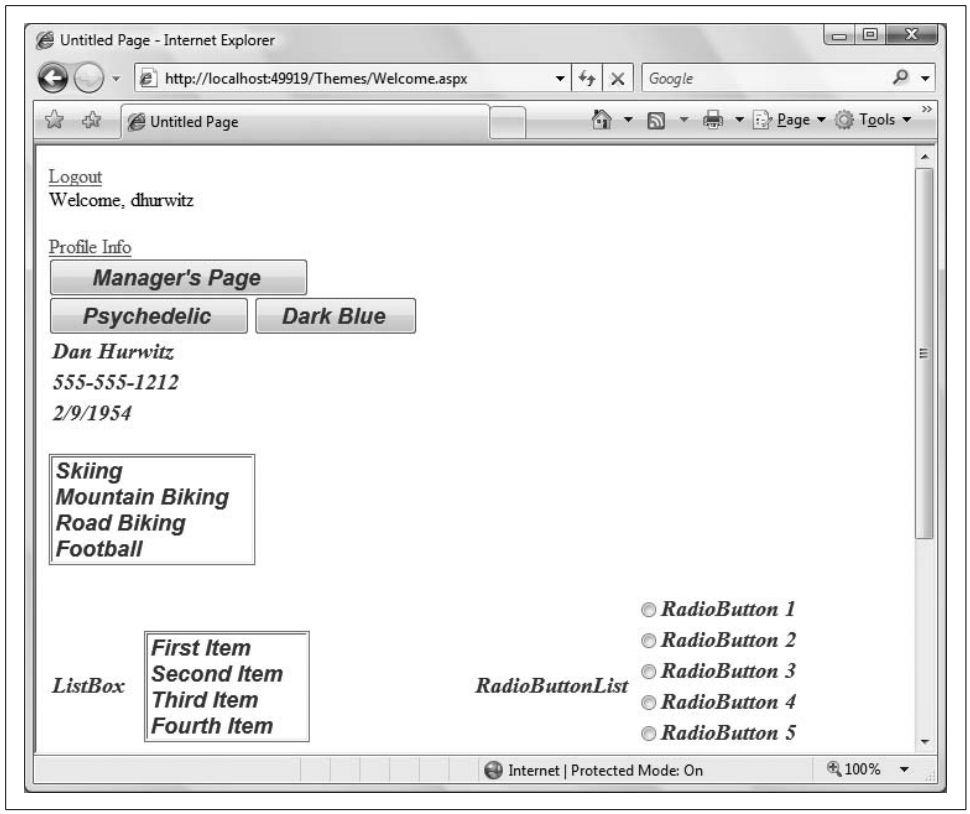


Figure 10-18. *The two buttons to set the page theme only become visible once the user has logged in. When the logged-in user chooses a theme, the theme is saved in the user's profile, and will be used every time that user logs in.*

Using Named Skins

You can override the theme for particular controls by using *named skins*. For example, you can set the `lblRadioButtonList` label to be red even in the Dark Blue theme by using a named skin. To accomplish this, create two Label skins in the *Label.skin* file within the *Dark Blue* folder:

```
<asp:Label Runat="server"
    ForeColor="Blue" Font-Size="Large"
    Font-Bold="True" Font-Italic="True" />
<asp:Label Runat="server" SkinID="Red"
    ForeColor="Red" Font-Size="Large"
    Font-Bold="True" Font-Italic="True" />
```

The first skin is the default; the second is a named skin because it has a `SkinID` property set to `Red`. Click the `lblRadioButtonList` control (that is, the `Label`, not the `RadioButtonList` itself) on the Welcome page in Design view and set the `SkinID`

property to Red (you may need to switch the Properties window back to properties from events). Or, open the source for *Welcome.aspx* and find the lblRadioButtonList and add the attribute SkinID="Red":

```
<asp:Label ID="lblRadioButtonList" Runat="server" Text="Radio Button List"
    SkinID="Red"/>
```

When you log in and set your theme to Dark Blue, you'll find that the label for the RadioButtonList is Red.

Summary

- Personalization allows your users to choose certain aspects of the site, which persist from one visit to the next. Personalization options can include your site's appearance, personal information, shopping carts, and other choices.
- A simple way to provide personalization is to define a profile for each user and store their personalization choices there.
- To use profiles in your site, modify *web.config*, create a <profile> section, and set the enabled property to true. Then, create a <properties> section within the <profile> section, and use the add attribute to add the names of any profile information you want to store.
- The default storage type for a profile property is string, but you can specify a different type with the Type attribute.
- The Profile.IsAnonymous property is a Boolean that lets you take action based on whether the user is logged in (using an anonymous profile).
- If you want users to be able to access the restricted features of your site without logging in, you can enable anonymous personalization. Simply add a line to *web.config* setting <anonymousIdentification enabled="true"/>.
- When the user logs in, if you want to copy the user's profile from the anonymous profile to the user's permanent profile, you need to handle the MigrateAnonymous event, which is automatically raised when the user logs in. The code for your event handler should copy the properties from the anonymous profile to the logged-in profile.
- You can allow users to customize the visual appearance of your site with themes. A theme is made up of skins, each of which provides the visual appearance of a single type of control.
- Themes are stored in a folder within your project named App_Themes. Each subfolder within App_Themes is the name of a specific theme. Within the named theme folders, you store specific skin files, one for each control you want to modify.
- A skin file looks like a control declaration, but it has no ID property, just appearance properties.

- To enable users to select their preferred theme on your site, add `<addname="Theme"/>` to the `properties` element in the `profile` section of *web.config*.
- To specify a customization theme on your page, just add a `Theme` attribute to the `Page` directive.
- To set the style sheet theme programmatically, you need to add an `Overrides Property StyleSheetTheme( )` method to the class for the page, and set the accessors appropriately.
- To set a customization theme programmatically, you need to add a `PreInit` event handler so that the theme is specified before the controls are created.
- To specify the theme for a specific control, use named skins by setting a `SkinID` property in the skin file.

You've now tried out just about every section of the ASP.NET toolbox, and you've seen controls ranging from the simple to the complex. You've learned how controls work together, and how to create event handlers for them. You know how to create sites that use master pages, allow users to log in, provide navigation tools, access databases, and do it all while using Ajax to enhance the experience. What's left to do? It's time to put it all together, of course. In the next chapter, you'll find a project that uses nearly everything you've learned so far to build a fully functional web site.

Quiz

1. What do you need to do to enable user profiles in your site?
2. What modifications do you need to make to *web.config* to specify what profile information you want to retain for your users?
3. What property of the `Profile` object do you use to determine whether a user is logged in?
4. How do you retain profile information that's not saved as a string?
5. How do you allow for profile information for a user without logging in?
6. How do you indicate that a specific piece of profile information should be retained for an anonymous user?
7. When a user logs in, how do you transfer any personalization data that the user might have entered as an anonymous user?
8. What's the difference between style sheet themes and customization themes?
9. Where do you specify settings for a skin?
10. How do you specify what theme to use on a page?

Exercises

Exercise 10-1. In this set of exercises, you're going to create a web site that hosts discussion forums for tropical fish enthusiasts. You won't actually write the code for the forums, but you'll build the framework so that users can log in if they wish and customize the site. Start by creating a site that has a front page welcoming users to the site and stating their login status. Add a login page, and a separate page where users can create their own accounts. Create a few users by whatever method you like to test these pages. The page for a logged-in user would look like Figure 10-19. (Hint: be sure to change the authentication method in *web.config*.)

Add a page to the site where users can customize their profiles. Add a `TextBox` to have users enter their preferred name. Add another `TextBox` where they can enter how many fish they have. Add a radio button group where users can choose between tropical or freshwater fish. Give them a `DropDownList` where they can choose their favorite fish type. The Profile page should look something like Figure 10-20 when it's filled in.

On the home page, provide an opportunity for logged-in users to edit their profile, and also display the contents of the profile on the home page. The home page should look something like Figure 10-21 when the user is logged in.

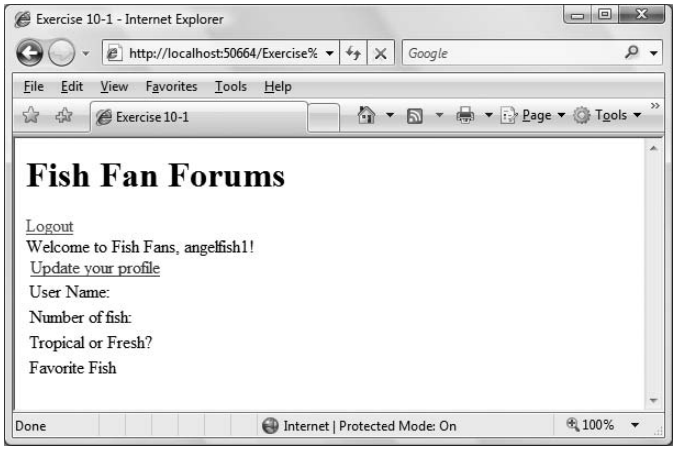


Figure 10-19. *A logged-in user page for Exercise 10-1.*

Figure 10-20. *The Profile page for Exercise 10-1.*

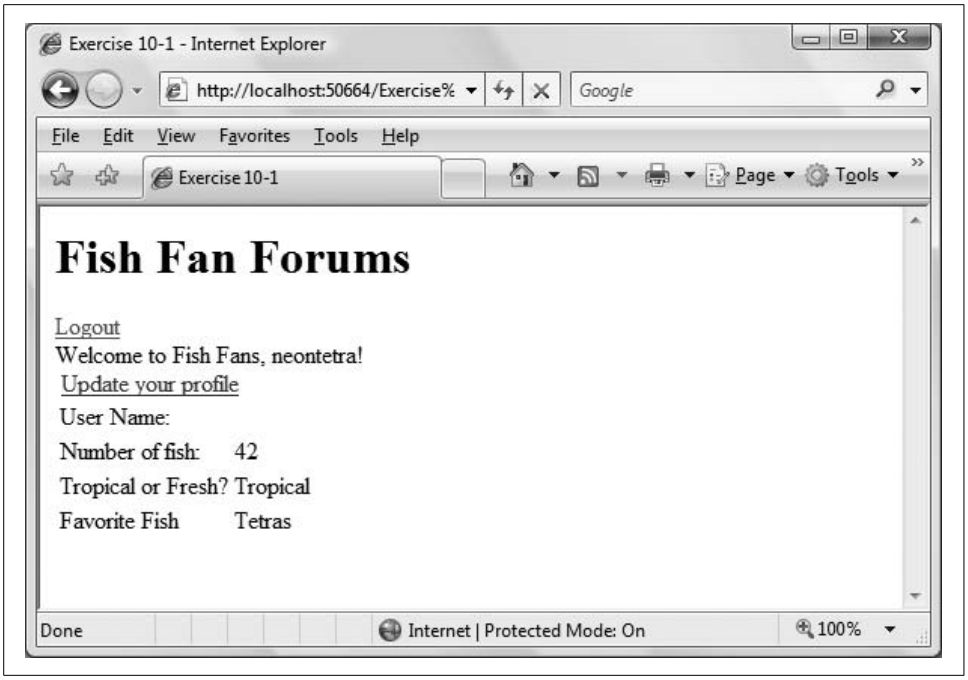


Figure 10-21. *The home page for Exercise 10-1.*

Exercise 10-2. Allow anonymous users to visit your site and create a temporary profile. They should be able to enter their preference for tropical or freshwater fish, but because they're anonymous, they shouldn't be able to choose a preferred username, or the number of fish they own. Enable the user to migrate their preference data from the anonymous profile to a registered, once they've created a user account.

Exercise 10-3. A page about tropical fish isn't very much fun unless it's colorful, but users probably want to color their pages to match their favorite fish. We'll keep it simple for this exercise: just create two theme files, with skin files for labels and buttons. Create the two theme files as follows:

- Angelfish theme: for buttons, set the foreground to yellow, the background to black, and the text to large. For labels, set the foreground to white, the background to black, and the text to large.
- Clownfish theme: for both buttons and labels, set the foreground to white, the background to orange, and the text to large.

Add a section to the user profile that lets users pick the theme they want to use, and have it displayed on all the pages of the site. The front page should look like Figure 10-22 for a user who has chosen the Angelfish theme.

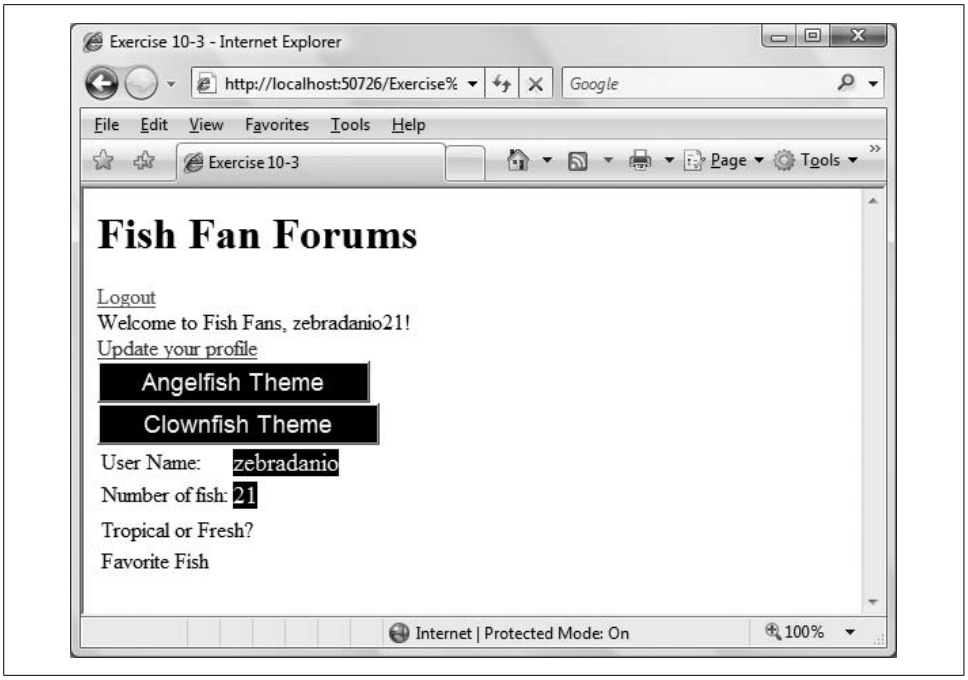


Figure 10-22. *Your goal for Exercise 10-3.*

CHAPTER 11

Putting It All Together

You've now practiced using just about every major tool in the ASP.NET toolbox. You've made dozens of sample applications, and you've gotten a feel for just how easy it is to make functional web sites with just a few controls. Now it's time to put those skills to the test. In this chapter, you'll make a fully functional (if somewhat limited) shopping application for the Adventure Works company. Unlike the order form you made in Chapter 2, this application will use all the skills you've learned. It uses data controls to display the Adventure Works database and retrieve the content the user wants, done in AJAX to speed things along. It has a shopping cart to store the items the user has purchased. It uses session state to pass that information on to a purchasing page. It incorporates validation controls to make sure the user enters good data. It has master pages that provide a consistent look and feel to the site, and custom error pages in case of problems. Finally, it has login controls to ensure that only registered users can access the pages of the site. In short, it's a fully functional working application.

Getting Started

Create a new web site titled All Together. This is the site that you'll use throughout the example in this chapter. This chapter consists of a single large example. As we build up the example, we will provide full code listings and shorter snippets along the way. At the end of the chapter are complete code listings for the entire example so you can see how everything fits together.

You can also download this example, as well as all of the other examples in this book, from *http://www.oreilly.com/catalog/9780596518455*.

Add a Master page to the web site, *MasterPage.master*. Be sure that the "Place Code in separate file" checkbox is checked.

Close the *Default.aspx* file and then delete it; you won't need it.

Add an Images folder to the web site by right-clicking on the root folder in the Solution Explorer and selecting New Folder. Insert a folder and call it *Images*.

This next step is really important: Create the logo file *Adventure Works Logo-250x70.gif* using any image-editing tool you like (our logo file is 250 pixels wide×70 pixels high), or download it from this book's web site. Once the image file is on your machine, it must be added to the project. Right-click on the Images folder and select Add Existing Item. Then, navigate to the logo file, wherever it is on your file system, and select it. It will automatically be copied to the Images folder and added to the project.

Adding Styles

You'll be using CSS styles for the various parts of your site, so you need to define the styles first. Add a CSS style sheet to the web site by selecting Website → Add New Item, and selecting Style Sheet. You can keep the default name of *StyleSheet.css*.

Copy in the styles from Example 11-1 any way you wish. You can type the styles in directly in the source code editing surface, or use the CSS editing tools described back in Chapter 6.

Example 11-1. *StyleSheet.css*

```
body
{
   font-family: Arial; Helvetica; sans-serif;
}
.ButtonSelect
{
   font-weight: normal;
   font-size: x-small;
   background-color: Yellow;
   color: Blue;
}
.ButtonText
{
   font-weight: bold;
   font-size: x-small;
   color: Black;
}
.Hyperlink
{
   font-weight: normal;
   font-size: small;
   color: Blue;
   text-decoration: underline;
}
.LabelMedium
{
```

Example 11-1. *StyleSheet.css (continued)*

```
   font-weight: bold;
   font-size: Medium;
   color: Black;
}
.LabelSmall
{
   font-weight: bold;
   font-size: small;
   color: Black;
}
.ListHeading
{
   font-weight: bold;
   text-decoration: underline;
   font-size: x-small;
   color: Black;
}
.MenuText
{
   font-weight: normal;
   font-size: small;
   color: Blue;
}
.PageTitle
{
   font-weight: bold;
   font-size: xx-large;
   color: Green;
}
.PageSubTitle
{
   font-weight: bold;
   font-size: x-large;
   color: Blue;
}
.TableCells
{
   font-weight: normal;
   font-size: small;
   color: Black;
   text-align: left;
   vertical-align: top;
}
.TableColumnHeading
{
   font-weight: bold;
   text-decoration: underline;
   font-size: small;
   color: Black;
   text-align: left;
}
.TableColumnHeadingRight
```

Example 11-1. *StyleSheet.css (continued)*

```
{
   text-align: right;
}
.TableNumberDecimal
{
   font-weight: normal;
   font-size: small;
   color: Black;
   text-align: right;
}
.TableRowHeading
{
   font-weight: bold;
   text-decoration: none;
   font-size: small;
   color: Black;
   text-align: left;
}
.TextBold
{
   font-weight: bold;
   font-style: italic;
   font-size: medium;
   color: Black;
}
.TextNormal
{
   font-weight: normal;
   font-size: medium;
   color: Black;
}
.TextSmall
{
   font-weight: normal;
   font-size: small;
   color: Black;
}
.TextXSmall
{
   font-weight: normal;
   font-size: x-small;
   color: Black;
}
.ValidationError
{
   font-weight: normal;
   font-size: small;
}
.Warning
{
   font-weight: bold;
   font-size: Small;
   color: Red;
```

Example 11-1. *StyleSheet.css (continued)*

```
}
.WarningRoutine
{
   font-weight: normal;
   font-size: Small;
   color: Red;
}
```

Using Master Pages

Add a new page, *Login.aspx*. Check both checkboxes: "Place code in separate file" and "Select master page." When the Master Page dialog comes up, select *MasterPage.master*. Remove the `Content` control that refers to the `ContentPlaceHolder` control named `head`.

Add several other new pages: *Home.aspx, Products.aspx, Cart.aspx, Purchase.aspx*, and *Confirm.aspx*. For each of these, select the same master page. Edit each of these pages to remove the `Content` control referring to the `head ContentPlaceHolder`. Set *Home.aspx* to be the startup page.

Open *MasterPage.master*. Add a style statement to the `<head>` element to import the style sheet, as in the highlighted line in the following snippet:

```
<head runat="server">
    <title>Untitled Page</title>
    <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
```

Delete the `ContentPlaceHolder` control with the ID of `head`.

Add an HTML table for layout, inside the `<div>` element, but before the content placeholder control. You can use the IDE tools or just type it manually in the Source view window. With the help of IntelliSense, I find it easier to type it manually.

Add the table structure and server controls below (above the `ContentPlaceHolder`):

```
<table border="0">
   <tr>
      <td colspan="4">
         <table>
            <tr>
               <td width="10px">
                   
               </td>
               <td>
                  <asp:ImageButton ID="ibLogo" runat="server"
                     ImageUrl="~/images/AdventureWorksLogo-250x70.gif"
                     AlternateText="AdventureWorks logo"
                     PostBackUrl="~/Home.aspx" />
               </td>
               <td width="10px">
                   
               </td>
```

```
                    <td width="500px" align="right">
                        <span class="PageTitle">Adventure Works</span>
                        <br />
                        <asp:Label ID="lblPageSubTitle" runat="server"
                            CssClass="PageSubTitle" Text="Page SubTitle" />
                        <br />
                        <asp:Label ID="lblTime" runat="server"
                            CssClass="TextXSmall" />
                    </td>
                    <td width="10px">
                         
                    </td>
                </tr>
                <tr>
                    <td colspan="5">
                        <hr />
                    </td>
                </tr>
            </table>
        </td>
    </tr>
</table>
<asp:contentplaceholder id="ContentPlaceHolder1" runat="server" >
</asp:contentplaceholder>
```

This code defines the table that you're going to use to hold the content of the master page. The first cell contains an `ImageButton` control to hold the logo for the site; when users click on the logo, it will take them to the *Home.aspx* page. The control displays the logo image file you created or downloaded earlier.

The cell to the right of the logo contains some text for the title and a pair of labels. Note the use of the `<span>` element on the page title; this allows you to apply a CSS class to it. The first label will contain the page subtitle, which will change depending on the page the user is on. The other label will contain the date and time, just for convenience.

Also note the use of the `border="0"` attribute in the opening `<table>` tag. This is a vestige of the development process. Although you might not want borders in the finished site, it is often helpful to make the cell borders visible during development by setting the border thickness to 1 pixel with `border="1"`. Then, when you are satisfied with the layout, set the borders back to 0 so they are no longer visible.

You'll need to populate the `Label` that shows the time, so open *MasterPage.master.vb*, the code-behind file for the master page. Create an event handler for the `Page_Load` event by selecting `(PageEvents)` from the left drop-down menu and `Load` from the right drop-down menu. Enter the following line of code:

```
lblTime.Text = DateTime.Now.ToString( )
```

Open *Home.aspx*. Edit the `Page` directive at the top of the file to set the `title` attribute to Home Page; also add the `trace` attribute at this time, but set it to false.

You'll need this because you know you are going to want to turn trace on or off during various phases of development.

Add a `MasterType` directive to the file also. This will enable the content page to access properties declared in the master page: if you haven't already, delete the `Content` control which refers to the `Head ContentPlaceHolder` that we deleted above. The complete markup for *Home.aspx* should look like the following:

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
    CodeFile="Home.aspx.vb" Inherits="Home" title="Home Page" Trace="false"%>
<%@ MasterType TypeName="MasterPage" %>
<asp:Content ID="Content2 ContentPlaceHolderID="ContentPlaceHolder1"
    Runat="Server">
</asp:Content>
```

Run the site now, to see what you've done so far. You should see something like Figure 11-1.

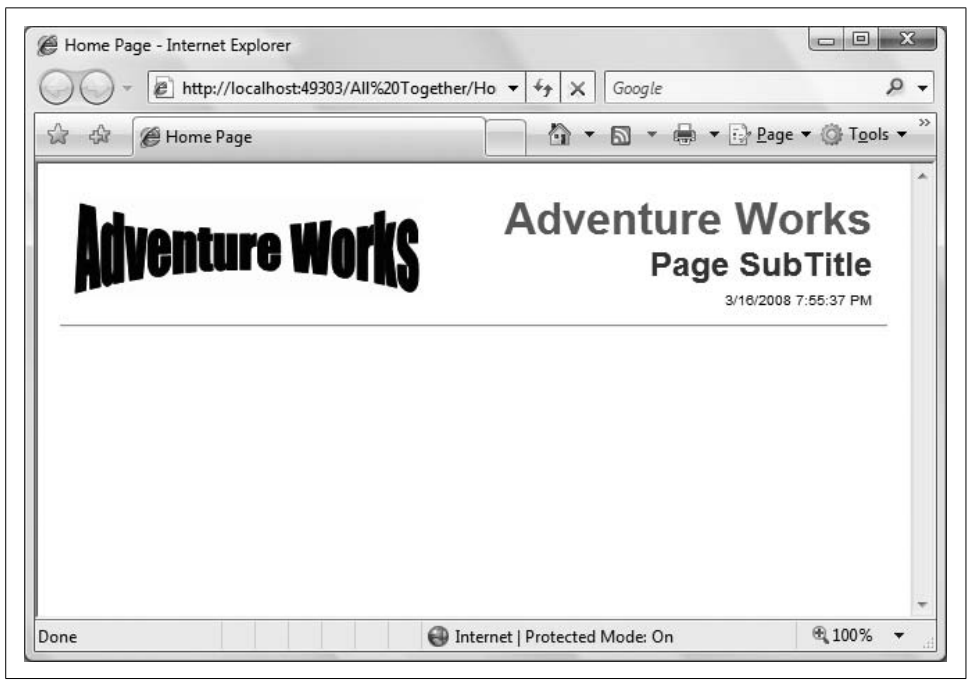


Figure 11-1. *Here's how the home page looks with nothing on it and the page subtitle not yet set.*

You want the page subtitle to display the current page; for example, Home or Products. The label is already in place in the master page.

Go to the code-behind file for the master page. Add the following code outside the `Page_Load` method and inside the class definition (the line that says `Partial Class MasterPage`) to create a public property called `PageSubTitle`, of type `Label`. (IntelliSense

can be a big help here. To insert a good starting code snippet, click on Edit → IntelliSense → Insert Snippet → Code Patterns → Properties, Procedures, Events → Define a Property.)

```
Public Property PageSubTitle( ) As Label
    Get
        Return lblPageSubTitle
    End Get
    Set(ByVal value As Label)
        lblPageSubTitle = value
    End Set
End Property
```

Then, in the code-behind of the home page, create a `Page_Load` method with the following highlighted line of code:

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    Me.Master.PageSubTitle.Text = "Home"
End Sub
```

If the IDE draws squiggly lines indicating some sort of problem, try building the web site by clicking on the Build menu item and selecting Build Web Site. The spurious error indicators will go away.

Switch over to the Source view of the home page and add some content inside the `Content` control, such as listed in Example 11-2.

Example 11-2. *Markup for the home page - Home.aspx*

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
    CodeFile="Home.aspx.vb" Inherits="Home" title="Home Page" Trace="false"%>
<%@ MasterType TypeName="MasterPage" %>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    <h2>This Is Your Home Page</h2>
    <div class="TextNormal">
        You can put some stuff about your company here. Perhaps some links.
        Of course, in a real world application, the navigation would probably
        much more complex. Also, the buttons would actually do something,
        rather than just wave their arms and say <span class="TextBold">Look
        at me!</span>
    </div>
</asp:Content>
```

Running the site now results in a page that looks like Figure 11-2.

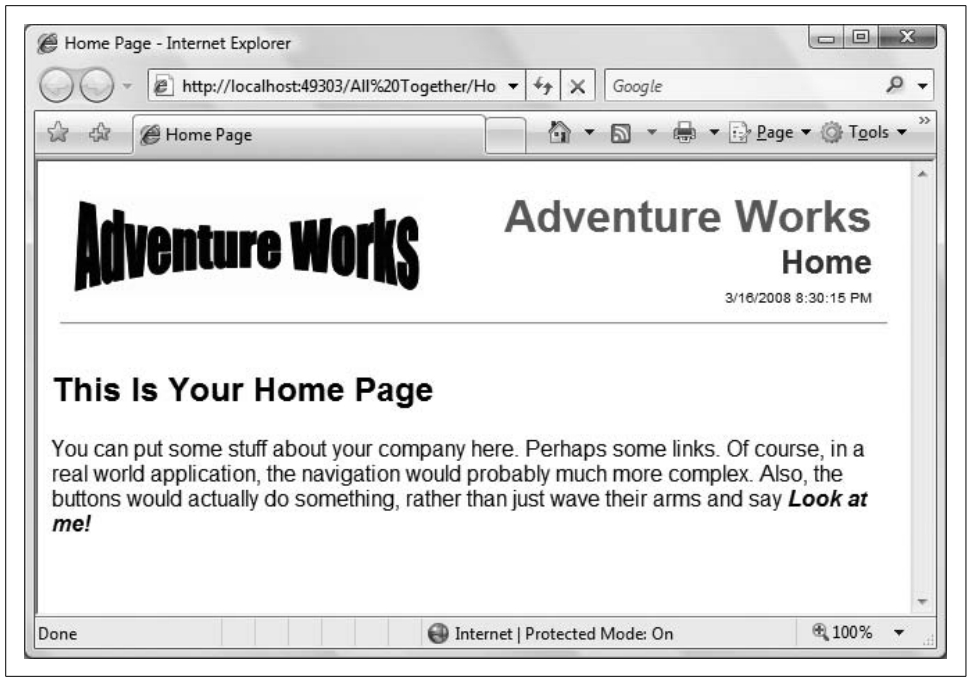

Home Page - Internet Explorer
http://localhost:49303/All%20Together/Ho
Google
Home Page
Page
Tools
Adventure Works
Adventure Works
Home
3/16/2008 8:30:15 PM
This Is Your Home Page
You can put some stuff about your company here. Perhaps some links. Of course, in a real world application, the navigation would probably much more complex. Also, the buttons would actually do something, rather than just wave their arms and say Look at me!
Done
Internet | Protected Mode: On
100%

Figure 11-2. *The home page now has the subtitle set and some content added.*

Setting Up Roles and Users

Your page has a good foundation, but you should add a measure of security to it to separate the customers from the managers. The next step is to enable security and then create a few users for your site.

Go to the WAT by selecting Website → ASP Configuration. Click on Security, and then click on "Select authentication type" under the Users column. Because this site will be available on the Internet, forms-based security is the way to go. Change the Authentication type to Forms by selecting "From the internet." You'll be setting up some roles to group the Adventure Works users into customers, employees, and managers. Back on the Security page, click on Enable roles. The security page should now look something like Figure 11-3.

Click on "Create or Manage roles", and create three roles: Manager, Employee, and Customer.

Click on the Back button to go back to the Security page. Click on Create user, and create four users, as in the following table.

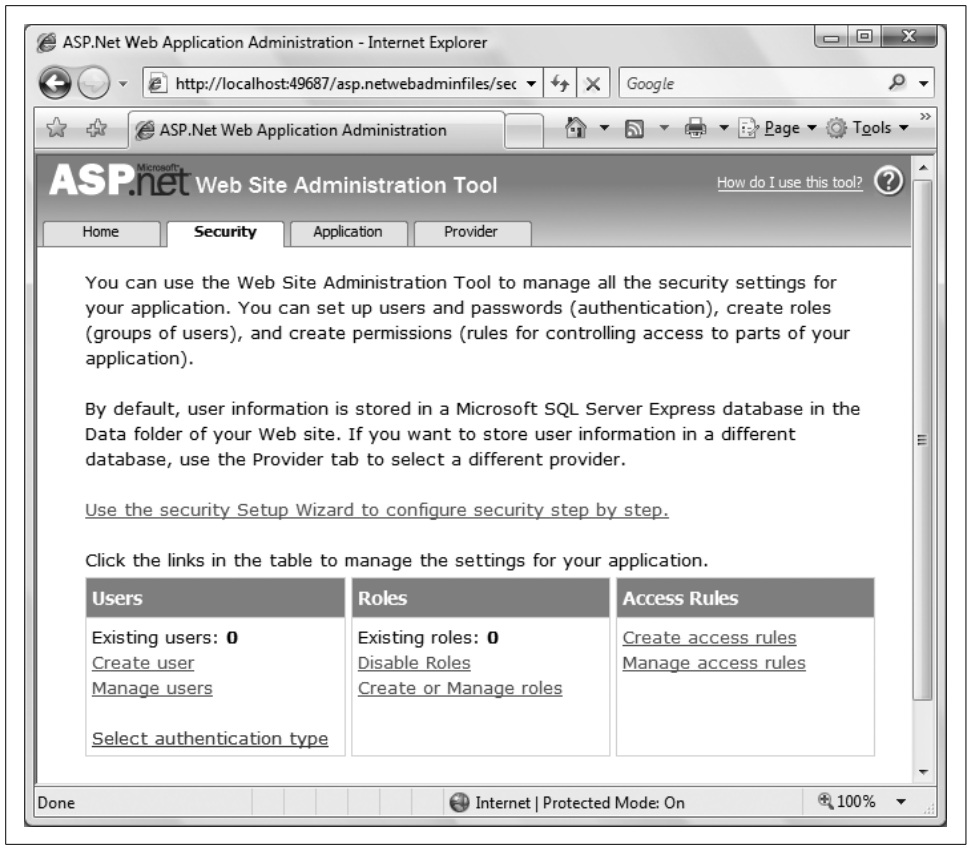

ASP.Net Web Application Administration - Internet Explorer
http://localhost:49687/asp.netwebadminfiles/sec
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).
By default, user information is stored in a Microsoft SQL Server Express database in the Data folder of your Web site. If you want to store user information in a different database, use the Provider tab to select a different provider.
Use the security Setup Wizard to configure security step by step.
Click the links in the table to manage the settings for your application.
Users
Existing users: 0
Create user
Manage users
Select authentication type
Roles
Existing roles: 0
Disable Roles
Create or Manage roles
Access Rules
Create access rules
Manage access rules
Done
Internet | Protected Mode: On
100%

Figure 11-3. *You've switched to Forms authentication, and enabled roles for your site, but there aren't any users just yet.*

User	Password	Role
bmacdonald	brian123!	Customer
dhurwitz	dan123!	Employee
jliberty	jesse123!	Customer
rhampster	rich123!	Manager

You must also provide an email address and a security question and answer for each user. We will not be using that information in this example, so it does not matter what you enter. Close the WAT. Now you have four users to work with for this example.

Logging In

Now that you have your users, you need a way for them to log in. Edit the master page markup file to add some login functionality.

Add another table row to the layout table, listed in Example 11-3. Note that the `ContentPlaceHolder` control has been moved to within one of the table cells.

Example 11-3. *Code snippet from MasterPage.master containing the login controls*

```
<tr>
    <td width="5px"> </td>
    <td width="150px" valign="top">
        <asp:LoginStatus ID="LoginStatus1" runat="server" CssClass="Hyperlink" />
        <br />
        <asp:LoginView ID="LoginView1" runat="server" >
            <LoggedInTemplate>
                <span class="WarningRoutine">Welcome</span>
                <asp:LoginName ID="LoginName1" runat="server"
                    CssClass="WarningRoutine"/>
            </LoggedInTemplate>
            <AnonymousTemplate>
                <span class="WarningRoutine">You are not logged in.
                     Please click the login link to log in to this website.</span>
            </AnonymousTemplate>
        </asp:LoginView>
    </td>
    <td width="5px"> </td>
    <td width="700px" valign="top" bgcolor="yellow">
        <asp:contentplaceholder id="ContentPlaceHolder1" runat="server" >
        </asp:contentplaceholder>
    </td>
</tr>
```

This code adds a new row to the table on the master page. The first and third cells are just spacers. The second cell holds a `LoginStatus` control and a `LoginView` control to go with it. Notice that the `CssClass` properties of both controls have been set to apply styles to them. The `LoginView` control has text added to it to present appropriate messages to either logged-in or anonymous users.

The fourth cell in the row now holds the `ContentPlaceHolder` control, so be sure to move the `ContentPlaceHolder` control that was outside the table to this cell.

Edit the *Login.aspx* page that you created earlier. Set the title in the `Page` directive, if you haven't already, remove the extra `Content` control, and add the same `MasterType` directive that you added to the home page. Drag a `Login` control into the Content area. Switch to Design view, click on the Smart Tag of the `Login` control, and click on Auto Format. Select the "Professional" scheme. Set the `DestinationPageUrl` property to `~/Home.aspx` so that users will be returned to the home page after they log in. You will end up with something like Example 11-4 for the markup for the Login page, with the changes highlighted.

Example 11-4. *Login.aspx*

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
    CodeFile="Login.aspx.vb" Inherits="Login" title="Login" Trace="false" %>
<%@ MasterType TypeName="MasterPage" %>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    <asp:Login ID="Login1" runat="server" DestinationPageUrl="~/home.aspx"
            BackColor="#F7F6F3" BorderColor="#E6E2D8" BorderPadding="4"
            BorderStyle="Solid" BorderWidth="1px" Font-Names="Verdana"
            Font-Size="0.8em" ForeColor="#333333">
        <TitleTextStyle BackColor="#5D7B9D" Font-Bold="True"
            Font-Size="0.9em" ForeColor="White" />
        <InstructionTextStyle Font-Italic="True" ForeColor="Black" />
        <TextBoxStyle Font-Size="0.8em" />
        <LoginButtonStyle BackColor="#FFFBFF" BorderColor="#CCCCCC"
            BorderStyle="Solid" BorderWidth="1px"
            Font-Names="Verdana" Font-Size="0.8em" ForeColor="#284775" />
    </asp:Login>
</asp:Content>
```

When you use the AutoFormat feature of a Smart Tag, it generally (but not always) uses hex values of red, green, and blue (RGB) to specify colors. You can also specify the colors with the common names, such as `Red`, `White`, and `Blue`.

Open the code-behind of the Login page. Create an event handler for the `Page_load` event and add the following highlighted line of code:

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    Me.Master.PageSubTitle.Text = "Login"
End Sub
```

All this does is set the page subtitle in the master page area.

Now run the site. You will see the first screen shown in Figure 11-4.

Click on the Login link to get the Login page, shown as the second screen in Figure 11-4. Enter the username and password for one of the user accounts you created earlier in this chapter. After you click on the Log In button, you will be brought back to the home page, shown as the third screen in Figure 11-4.

It is important that you actually click on the Log In button, rather than just pressing the Enter key. The Log In button does not have focus, the `ImageButton` on the master page displaying the logo does. So, if you press the Enter key, it will take you back to the home page without logging you in.

Earlier, you enabled roles in the WAT and added each user to one of the three roles: Manager, Customer, and Employee. As you saw in Chapter 10, you can use these roles to present customized content to the users who visit the page. You'll add two `Panel` controls to the home page that present content depending on the user's role.

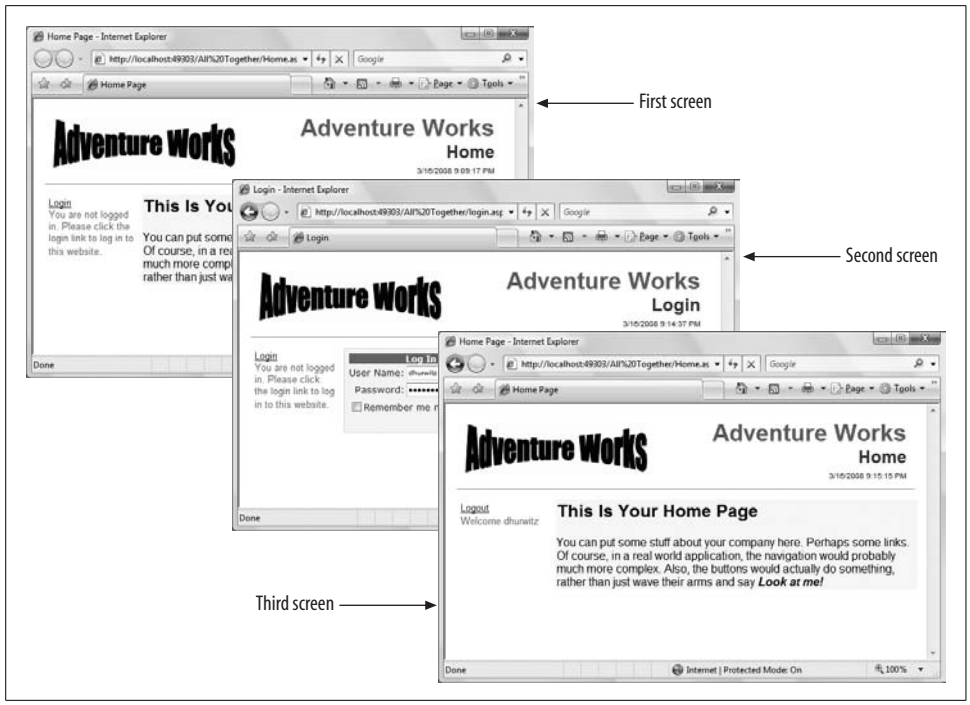

First screen
Second screen
Third screen
Adventure Works
Home
Login
Adventure Works
Login
Adventure Works
Home
This Is Your Home Page
You can put some stuff about your company here. Perhaps some links. Of course, in a real world application, the navigation would probably much more complex. Also, the buttons would actually do something, rather than just wave their arms and say Look at me!

Figure 11-4. *When you first run the site, you'll see the screen on the left. After you click the Login link, you'll be taken to the second screen, and after you've successfully logged in, you'll be taken back to the home page, which now looks like the third screen.*

Edit *Home.aspx* to see this in action. Add two `Panel` controls, as listed in Example 11-5, to the page, inside the `Content` control, after the closing `<div>` for the text that all users see.

Example 11-5. *Role-specific content in Home.aspx*

```
<asp:Panel ID="pnlEmployee" runat="server" Visible="false" >
    <h3>Employee Information</h3>
    <div class="TextNormal">
        This panel should only be visible to users are a members of the
        <b>Employee</b> role. Turning on the visibility of this Panel occurs in the
        Page_Load event handler.
    </div>
</asp:Panel>
<asp:Panel ID="pnlManager" runat="server" Visible="false" >
    <h3>Manager Information</h3>
    <div class="TextNormal">
        This panel should only be visible to users are a members of the
        <b>Manager</b> role. Turning on the visibility of this Panel occurs in the
        Page_Load event handler.
    </div>
</asp:Panel>
```

Switch over to the code-behind for the home page, *Home.aspx.vb*. Add the highlighted lines of code from Example 11-6 to the `Page_Load` event handler.

Example 11-6. *Controlling visibility based on roles in Home.aspx.vb*

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    Me.Master.PageSubTitle.Text = "Home"

    ' control the visibility of sections restricted to specific roles
    pnlManager.Visible = User.IsInRole("Manager")
    pnlEmployee.Visible = User.IsInRole("Employee")
End Sub
```

The code here is very simple; it sets the visibility of each panel depending on the value of the `IsInRole` method for the appropriate role.

Before logging in, the home page will still look like the first screen shown previously in Figure 11-4. If you log in as rhampster, who is a member of the Managers role (it's only fitting that the boss is a rodent), you will see Figure 11-5.

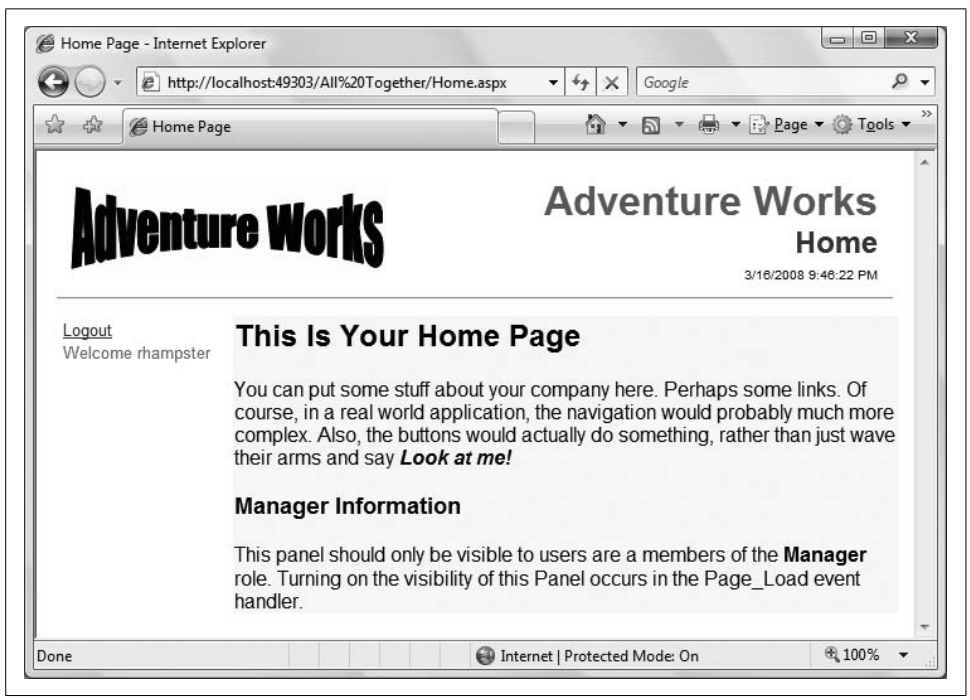

Home Page - Internet Explorer
http://localhost:49303/All%20Together/Home.aspx
Google
Home Page
Page
Tools
Adventure Works
Adventure Works
Home
3/16/2008 9:46:22 PM
Logout
Welcome rhampster
This Is Your Home Page
You can put some stuff about your company here. Perhaps some links. Of course, in a real world application, the navigation would probably much more complex. Also, the buttons would actually do something, rather than just wave their arms and say Look at me!
Manager Information
This panel should only be visible to users are a members of the Manager role. Turning on the visibility of this Panel occurs in the Page_Load event handler.
Done
Internet | Protected Mode: On
100%

Figure 11-5. *If you log in as a member of the Manager role, you'll see the manager-specific information.*

Now, log out and log in as user dhurwitz, and you'll see just the content of `pnlEmployee`. Log in again as user jliberty or bmacdonald, and you won't see either panel because customers don't need to see employee-specific information. Of course, if you make a user a member of both the Manager and Employee roles, she would see both panels.

Navigation

The front page of your site is looking pretty good. Users can identify themselves, and see the custom content. The master page is working as planned, and each page identifies itself appropriately. The next thing to do is add some navigation tools so that users can find their way around, which means you have to create a site map. Close the browser if it is open, and select Website → Add New Item, and choose Site Map. Accept the default name of *Web.sitemap*.

As you learned in Chapter 6, the site map is an XML file, and you have to create it manually—the IDE won't do it for you. Open the *web.sitemap* file, and replace the default boilerplate with the highlighted code in Example 11-7.

Example 11-7. *Web.sitemap*

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode title="Root" >
        <siteMapNode url="~/Home.aspx" title="Home" description="Home page" />
        <siteMapNode url="~/Products.aspx" title="Products"
            description="Products offered by AdventureWorks" />
        <siteMapNode url="~/Cart.aspx" title="Shopping Cart"
            description="Items selected for purchase" />
        <siteMapNode url="~/Purchase.aspx" title="Purchase"
            description="Purchase your selected items" />
    </siteMapNode>
</siteMap>
```

Now that you have the site map file, you'll add the navigation controls to the master page. Add the following code to *MasterPage.master* in the same table cell and after the `LoginView` control:

```
    <hr />
    <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server"
        ShowStartingNode="false" />
    <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1"
        CssClass="MenuText" />
```

In this case, you're using a `Menu` control rather than a `TreeView`. Note that the `DataSourceID` property of the `Menu` control points to the `SiteMapDataSource` control that you just created. You've set the control's `ShowStartingNode` property of the `SiteMapDataSource` control to `false` to suppress display of the root node in the menu.

You don't want anonymous users to be able to use the menu, so add the following code to the Page_Load method of *MasterPage.master.vb* to disable the menu if the user is not logged in:

```
If Page.User.Identity.IsAuthenticated = False Then
    Menu1.Enabled = False
End If
```

Anonymous users will be able to see the menu; they just won't be able to click anything on it.

At this point, a user could bypass the login by entering the URL of any of the other pages directly into the browser. To prevent this, add the following code to the Page_ Load method of every page in the web site except Home and Login (where you want to allow anonymous users):

```
If User.Identity.IsAuthenticated = False Then
   Response.Redirect("login.aspx")
End If
```

Go ahead and try out the site now to make sure everything is working. When you first see the home page, you'll see that the navigation menu is disabled. Try entering *cart.aspx* in the address field of your browser. You'll see that you're taken back to the Login page instead.

One other navigation aid that you can add is setting the page subtitle in the master page to identify to the user where they are. You have already done this for the Home and Login pages. For the rest of the pages, add the MasterType directive to the top of the relevant markup file:

```
<%@ MasterType TypeName="MasterPage" %>
```

While in each page markup file, also set the title attribute of the Page directive. Then, go to the Page_Load method of each page code-behind file and add a line similar to the following (for the Purchase page), which sets the page subtitle:

```
Me.Master.PageSubTitle.Text = "Purchase"
```

Most real-world sites would have a somewhat deeper menu structure, and so might benefit from a SiteMapPath control to provide bread crumbs.

Test out everything to see how it works.

Products Page

The intent of the Products page is to allow the user to select a product category and then see a grid displaying all the products in that category. The user can select any of these products to see more detail about that product, and if she wants, she can add that item to the shopping cart by clicking on a button. The finished page can be seen in Figure 11-6.

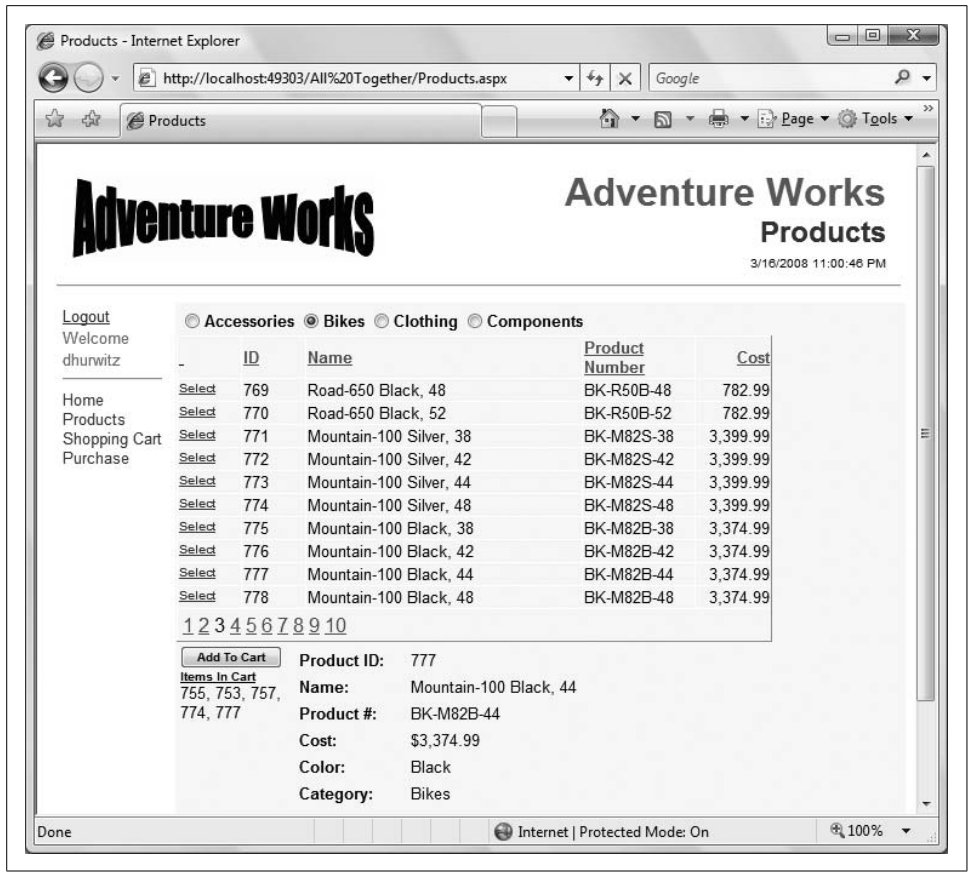


Figure 11-6. *The Products page is now finished, and has several items added to the cart.*

This page has several data-bound controls: a `RadioButtonList` for selecting the product category, a `GridView` for displaying the products (filtered by category), and a `DetailsView` for displaying details about the currently selected item.

In addition, visibility of some of the controls is turned off, depending on circumstances. Initially, only the `RadioButtonList` is visible. Once the user has selected a category, the `GridView` is made visible. When the user selects a product from the `GridView`, the `DetailsView` and its associated button for adding the item to the cart are made visible.

Open *Products.aspx*. You will start with a data source control. Go to Source view, if not there already, and drag a `SqlDataSource` control from the Data section of the Toolbox into the `Content` control on the page. Set the control's ID to `sqlCategories`. Switch to Design view and click on the Smart Tag. Configure it to point to the Adventure Works database—if you did the examples in Chapter 4, you may still have a data connection set up that refers to *AdventureWorks.mdf*. If not, you should probably flip back to Chapter 4 and review the section on creating a database

connection. In the "Configure the Select Statement" portion of the Wizard, select the "Specify a custom SQL statement" radio button. After you click Next, enter the following custom statement:

```
select Name, ProductCategoryID from Production.ProductCategory order by Name
```

You could have built this statement in the Wizard, but remember from Chapter 4 that the IDE doesn't automatically include the `Production` schema in the Select statement, so this custom statement is easier. Test the query to make sure everything is working, and finish the Wizard.

Drag a `RadioButtonList` control from the Standard section of the Toolbox onto the content section of the page. Set its ID to `rblCategories`. In Design view, click on its Smart Tag and select Choose Data Source. In the Data Source Configuration Wizard, select `sqlCategories` as the data source, `Name` as the data field to display, and `ProductCategoryID` as the data field for the value, as shown in Figure 11-7.

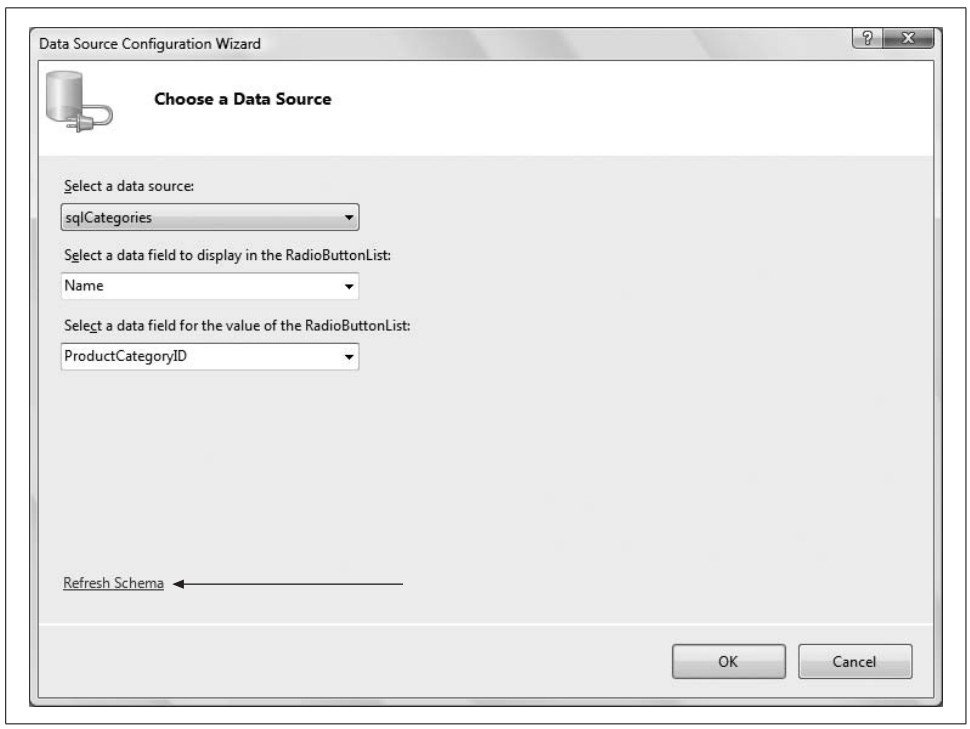


Figure 11-7. *If you can't see any of the data fields in the drop-down menu controls, click the "Refresh Schema" link to see them.*

If none of the fields are visible in the drop-down menus, click on the Refresh Schema link, indicated with the arrow in Figure 11-7.

Set the RepeatDirection property of rblCategories to Horizontal, and the AutoPostBack property to True, so that the page will post back as soon as a change is made (later you will add AJAX features to avoid the flicker), and the CssClass property to LabelSmall.

Run the web site, log in, and go to the Products page. You will see a set of four radio buttons, as shown in Figure 11-8.

Figure 11-8. *Once you've logged in and navigated to the Products page, you'll see the list of product radio buttons.*

Stop the application, and then drag another SqlDataSource control onto the content area to be the data source for the products grid. Set its ID to sqlProducts. This data source will return all the products of the category specified in the radio buttons, so you need to pass the value of the selected radio button to the data source as a parameter. Unfortunately, the Data Source Configuration Wizard shown previously in Figure 11-7 does not do parameterized queries, so you need to enter the code directly into Source view, as shown in the following code snippet:

```
<asp:SqlDataSource id="sqlProducts" runat="server"
    ConnectionString=
        "<%$ ConnectionStrings:AdventureWorksConnectionString %>">
    <SelectParameters>
        <asp:ControlParameter ControlID="rblCategories"
            Name="ProductCategoryID"
            PropertyName="SelectedValue" />
    </SelectParameters>
</asp:SqlDataSource>
```

The ConnectionString attribute points to the previously configured connection string. Your connection string may have a different name than that shown here. The SelectParameters element specifies that the parameter will be called ProductCategoryID and will come from the SelectedValue property of the rblCategories control.

But where is the SQL Select command, and where is this parameter used? You could declare a SelectCommand attribute, as you did for the first SqlDataSource, but this query is sort of long and complex, with a subquery as well as the parameter. So you will set the SelectCommand property of the control programmatically in the Page_Load of the Products page. Open the Page_Load method in *Products.aspx.vb*, and add the following code:

```
Dim strCommand As String = String.Empty
strCommand = "select ProductID, Name, ProductNumber, ListPrice from " + _
               "Production.Product "
strCommand += "where ProductSubcategoryID in "
strCommand += "(select ProductSubcategoryID from " + _
               "Production.ProductSubcategory "
strCommand += "where ProductCategoryID = "
strCommand += "@ProductCategoryID)"
sqlProducts.SelectCommand = strCommand
```

The parameter, @ProductCategoryID (highlighted in the above code snippet), assumes the value of the selected radio button. When the page first loads and none of the radio buttons are selected, this query returns nothing, so the GridView does not display. But as soon as a value is selected, the query returns rows and they display in the GridView.

To see this, drag a GridView control from the Data section of the Toolbox onto the content area. Set its ID property to gvProducts and its DataKeyNames property to ProductID. In Design view, click on its Smart Tag and set its Data Source to be sqlProducts. While the Smart Tag is open, check the Enable Paging, Enable Sorting, and Enable Selection checkboxes, as shown in Figure 11-9.

Click on the Edit Columns link in the Smart Tag, where you'll specify the columns from the SELECT query: ProductID, Name, ProductNumber, and ListPrice. Be sure to uncheck the Auto-generate fields checkbox. Although you want all the fields from the query to display, manually adding the columns to the GridView allows you to fully specify the appearance and behavior of each column.

To add each field from the query, make sure BoundField is selected in the Available fields list, and then click the Add button. For the first column, set the DataField for this BoundField to ProductID, the SortExpression to ProductID, the HeaderText to ID, and the ItemStyleWidth to 50px, as shown in Figure 11-10. Then, add each of the other columns accordingly.

Alternatively, you can declare all the fields directly in Source view, or any combination of techniques that works for you. In any case, you should end up with the

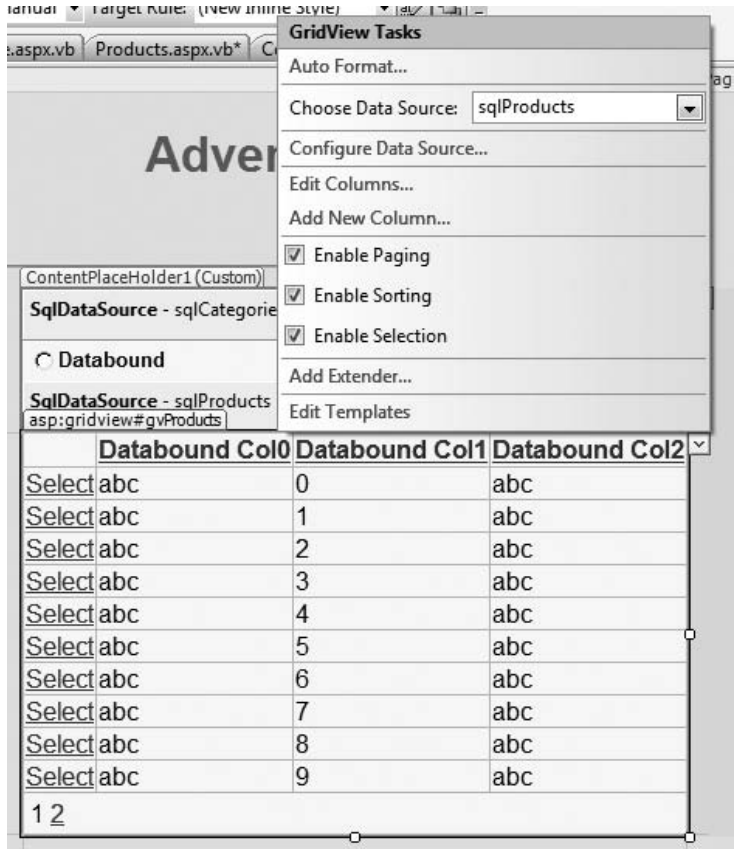


Figure 11-9. *After you've selected the data source, and enabled Paging, Sorting, and Selection, the gvProducts GridView will look like this.*

following declaration for the products `GridView`, including several attributes of the `GridView` itself and all the columns within the `<Columns>` element:

```
<asp:GridView id="gvProducts" runat="server"
    DataSourceID="sqlProducts" DataKeyNames="ProductID"
    AllowSorting="True" AllowPaging="True"
    AutoGenerateColumns="False"
    HeaderStyle-CssClass="TableColumnHeading"
    RowStyle-CssClass="TableCells">
    <Columns>
        <asp:CommandField ShowSelectButton="True" ItemStyle-Width="50"
            ControlStyle-CssClass="ButtonSelect" />
        <asp:BoundField DataField="ProductID" HeaderText="ID"
            SortExpression="ProductID">
            <ItemStyle Width="50px" />
        </asp:BoundField>
        <asp:BoundField DataField="Name" HeaderText="Name"
```

```
            SortExpression="Name">
            <ItemStyle Width="225px" />
        </asp:BoundField>
        <asp:BoundField DataField="ProductNumber"
            HeaderText="Product Number"
            SortExpression="ProductNumber">
            <ItemStyle Width="90px" />
        </asp:BoundField>
        <asp:BoundField DataField="ListPrice" HeaderText="Cost"
            SortExpression="ListPrice"
            ItemStyle-CssClass="TableNumberDecimal"
            HeaderStyle-CssClass="TableColumnHeadingRight">
            <ItemStyle Width="60px" />
        </asp:BoundField>
    </Columns>
</asp:GridView>
```

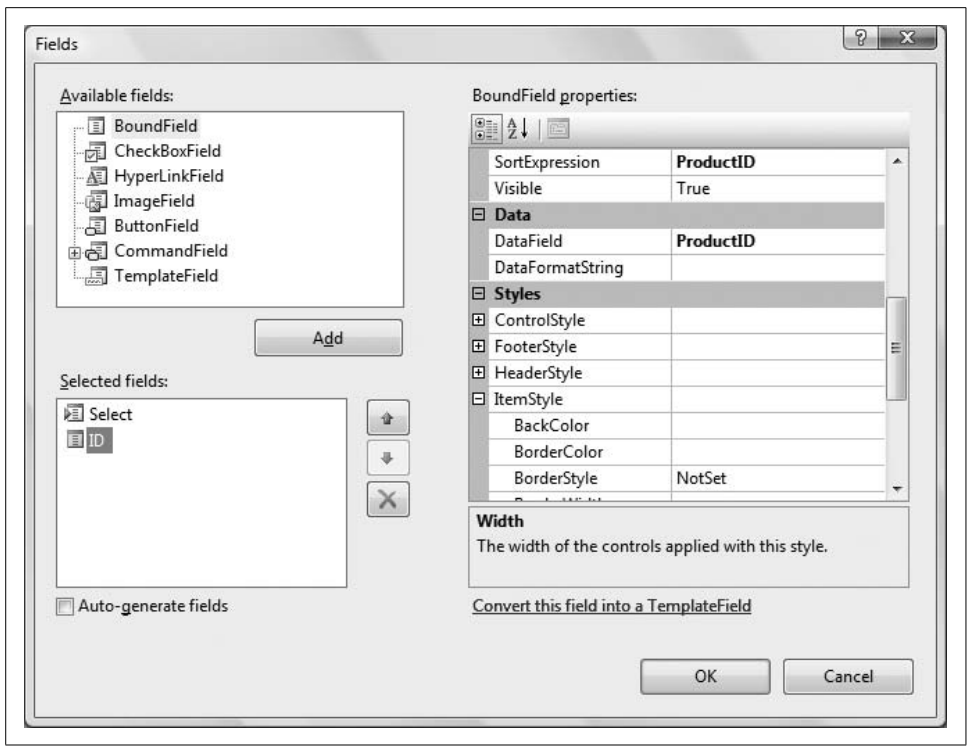


Figure 11-10. *Specify the ProductID bound field in the Fields dialog box.*

The `DataKeyNames` attribute is very important. It specifies the name (or names) of the field(s) that make up the primary key for the items displayed. In this example, the primary key is a single field, `ProductID`.

As you can see, there are many CSS-related attributes, all of which allow you to apply a style to a specific type of element in the grid.

Run the site, log in, navigate to the Products page, and select a category. You'll see that all the products for that category are listed in the grid.

Now you need to display the item details when the user selects an item from the grid. Drag a Panel control onto the page, inside the Content area but after `gvProducts`. Set its ID to `pnlProduct`. Inside the Panel is going to be a layout table with a `DetailsView` control data bound to another `SqlDataSource`.

We haven't used the `DetailsView` control previously in this book. It is a databound control, similar to the `GridView`, but it is used to display or edit a single record at a time. In this example, it displays the details about the single record selected from the `GridView`.

The contents of the `pnlProduct` are listed in Example 11-8.

Example 11-8. *Panel pnlProduct on Products page*

```
<asp:Panel id="pnlProduct" runat="server" Visible="false">
    <table width="100%">
        <tr>
            <td valign="top">
                <asp:Button id="btnAddToCart" runat="server"
                    Text="Add To Cart" OnClick="btnAddToCart_Click"
                    CssClass="ButtonText" />
                <div class="ListHeading">Items In Cart</div>
                <asp:Label ID="lblCart" runat="server" CssClass="TextSmall"
                    Width="90"/>
            </td>
            <td valign="top">
                <asp:SqlDataSource id="sqlDetailsView" runat="server"
                    ConnectionString=
                        "<%$ ConnectionStrings:AdventureWorksConnectionString %>">
                    <SelectParameters>
                        <asp:ControlParameter ControlID="gvProducts"
                            Name="ProductID"
                            PropertyName="SelectedDataKey.Values['ProductID']" />
                    </SelectParameters>
                </asp:SqlDataSource>
                <asp:DetailsView id="DetailsView1" runat="server"
                    DataSourceID="sqlDetailsView" DataKeyNames="ProductID"
                    AutoGenerateRows="false"
                    CssClass="TableCells" BorderWidth="0"
                    FieldHeaderStyle-CssClass="TableRowHeading"
                    CellSpacing="2" CellPadding="2" Width="500px" Height="50px">
                    <Fields>
                        <asp:BoundField DataField="ProductID"
                                    HeaderText="Product ID:"
                                    SortExpression="ProductID" />
                        <asp:BoundField DataField="Name" HeaderText="Name:"
                                    SortExpression="Name" />
                        <asp:BoundField DataField="ProductNumber"
                                    HeaderText="Product #:"
                                    SortExpression="ProductNumber" />
```

Example 11-8. *Panel pnlProduct on Products page (continued)*

```
                    <asp:BoundField DataField="ListPrice" HeaderText="Cost:"
                                SortExpression="ListPrice"
                                DataFormatString="{0:C}" HtmlEncode="false"/>
                    <asp:BoundField DataField="Color" HeaderText="Color:"
                                SortExpression="Color" />
                    <asp:BoundField DataField="CategoryName"
                                HeaderText="Category:"
                                SortExpression="CategoryName" />
                    <asp:BoundField DataField="SubcategoryName"
                                HeaderText="SubCategory:"
                                SortExpression="SubcategoryName" />
                    <asp:BoundField DataField="Description"
                                HeaderText="Description:"
                                SortExpression="Description" />
                </Fields>
            </asp:DetailsView>
        </td>
    </tr>
</table>
</asp:Panel>
```

As with the previous parameterized query, you will set the `SelectCommand` for the data source, `sqlDetailsView`, in the `Page_Load` of the Products page. In this case, the `ProductID` value of the row selected in `gvProducts` is passed as the parameter to the `SQLDataSource` named `sqlDetailsView`. Add the following code to the `Page_Load` method, after setting the `SelectCommand` property of the previous data source:

```
strCommand = String.Empty
strCommand += "select product.*, subcat.ProductSubcategoryID, " + _
              "subcat.Name as SubcategoryName, "
strCommand += "cat.ProductCategoryID, cat.Name as CategoryName, "
strCommand += "model.Name as ModelName, model.CatalogDescription, " + _
              "model.Instructions, "
strCommand += "description.Description "
strCommand += "from Production.Product product "
strCommand += "join Production.ProductSubcategory subcat on " + _
              "product.ProductSubcategoryID = subcat.ProductSubcategoryID "
strCommand += "join Production.ProductCategory cat on subcat.ProductCategoryID = " +_
              "cat.ProductCategoryID "
strCommand += "join Production.ProductModel model on product.ProductModelID = " + _
              "model.ProductModelID "
strCommand += "join Production.ProductModelProductDescriptionCulture culture on " + _
              "model.ProductModelID = culture.ProductModelID "
strCommand += "join Production.ProductDescription description on " + _
              "culture.ProductDescriptionID = description.ProductDescriptionID "
strCommand += "where product.ProductID = @ProductID and culture.CultureID = 'en' "
sqlDetailsView.SelectCommand = strCommand
```

Inside the Panel is also a `Button`, `btnAddToCart`. Switch to Design view and double-click the button to open the code-behind in the skeleton of an event handler, ready for you to type. The event handler code is included in Example 11-9. This method

SQL CHEAT SHEET

Joins

All of the queries you have seen in this book so far have been simple `SELECT` statements from a single table. The true strength of a relational database comes from using multiple tables to contain *normalized data*. Data that has been normalized essentially means there is no duplicate data.

Suppose you have a database containing employment information. Each employee has not only a job title but also a job description. Rather than have identical job descriptions in the `Employee` table for every employee with the same job, it is much better to have the job titles and descriptions in a separate `Jobs` table, and then refer to that `Jobs` record in the `Employees` table. There is said to be a *relationship* between the `Employees` table the `Jobs` table.

Now, however, when you want to query the data, you must join the two tables back together in your query statement. This is done with the SQL keyword `JOIN`.

The `JOIN` keyword alone, as used in the preceding snippet, is the default join type, known as an *inner join*. This means that any rows in either table that do not match the selection criteria will not be included in the results.

There are many circumstances where you do not want to omit these records, in which case you must use an *outer join*. There are several different types of outer joins, including left, right, cross, and full, depending on which data specifically you want to include and which to omit.

For a complete discussion on SQL queries in general and joins in particular, we highly recommend *Transact-SQL Programming* by Kevin Kline et al. (O'Reilly). Although this book is a bit dated, only covering up through SQL Server 7.0, the basic syntax has not changed, and this book remains an excellent primary reference for SQL programming.

retrieves the `ProductID` of the selected item using the `Value` of the `SelectedDataKey` property of the `GridView`. Then, it checks if the `Session` object exists, and in either case updates it with the currently selected item as a comma-separated string. It also displays the contents of the cart in a `Label` control. (The space trailing the comma allows the content of the Label control to wrap when many items are listed.)

While you're in the code-behind file, add the single-line event handler for the `SelectedIndexChanged` event of the grid `gvProducts`, also listed in Example 11-9. This displays the details of the selected item.

Also add an event handler to `gvProducts` for the `RowDataBound` event. This allows you to apply formatting to the cost display. There is an easier way to set the format in this case, which you will use later in the chapter, but this demonstrates a really powerful technique that comes in handy with almost every project. That technique involves looking at each row as it is bound to the data and applying some formatting

on a row-by row basis. It is even possible to make different formatting decisions based on the content of each row.

Finally, add an event handler for the `SelectedIndexChanged` of the `RadioButtonList` `rblCategories`, which hides the detail `Panel` when a new category is selected. This prevents the details of the previous item remaining displayed.

Example 11-9. *Products.aspx.vb event handlers*

```
Protected Sub btnAddToCart_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
    ' the contents of the cart will be saved in a Session object as
    '     a string of comma-delimited values of ProductID's
    Dim strCart As String = String.Empty
    Dim strProductId As String = gvProducts.SelectedDataKey.Value.ToString( )

    If Session("Cart") Is Nothing Then
        strCart = strProductId
    Else
        strCart = Session("Cart").ToString( ) + ", " + strProductId
    End If
    Session("Cart") = strCart
    lblCart.Text = strCart
End Sub

Protected Sub gvProducts_SelectedIndexChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles gvProducts.SelectedIndexChanged
    pnlProduct.Visible = True
End Sub

Protected Sub gvProducts_RowDataBound(ByVal sender As Object, _
        ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs) _
        Handles gvProducts.RowDataBound
    Dim str As String = String.Empty
    If e.Row.RowType = DataControlRowType.DataRow Then
        Dim cell As TableCell = e.Row.Cells(4) ' ListPrice cell
        Dim nCost As Decimal
        Try
            nCost = CType(cell.Text, Decimal)
            str = nCost.ToString("##,##0.00")
        Catch ex As ApplicationException
            str = "n.a."
        Finally
            cell.Text = str
        End Try
    End If
End Sub

Protected Sub rblCategories_SelectedIndexChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles rblCategories.SelectedIndexChanged
    pnlProduct.Visible = False
End Sub
```

The finished Products page, with several items added to the cart, is shown back in Figure 11-6.

Notice the nifty way that the `nCost` variable, which is a number, is converted to a string with the proper format, including a comma and a decimal point. The `ToString()` method allows you to pass in an optional *format string* to control the appearance of its output. The zeros are placeholders for required digits; the # symbols are placeholders for optional digits.

Adding AJAX

It is easy to spice up the performance of the Products page with a little help from AJAX. All you need to do is wrap the entire contents of the `Content` control inside an `UpdatePanel` control. You can do this by dragging an `UpdatePanel` control from the AJAX Extensions section of the Toolbox onto the page in Design view, and then dragging all the existing content inside the `UpdatePanel`. Alternatively, go to Source view and add the following highlighted lines of code, wrapping the content of the `Content` control:

```
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
        Runat="Server">
    <asp:UpdatePanel id="UpdatePanel1" runat="server">
        <ContentTemplate>

              ... all the content goes here ...
        </ContentTemplate>
    </asp:UpdatePanel>
</asp:Content>
```

The `ScriptManager` control should be on the master page, so you don't have to add one to every content page. So, open *MasterPage.master* and drag a `ScriptManager` control onto the page, if it is not already there. Run the page to make sure everything works.

Cart Page

The Cart page displays the contents of the cart and allows you to remove items from the cart. It also provides a button to purchase the items in the cart, which would, of course, take you to the Purchase page. The finished page can be seen in Figure 11-11.

Of course, a full-featured cart would provide much more functionality than the simple cart shown here. For example, your fully featured commercial site might use personalization to remember what was added to the cart in previous sessions and restore that information in a new session. It would almost certainly allow the user to change the quantity ordered of a given item, not to mention things such as size or color.

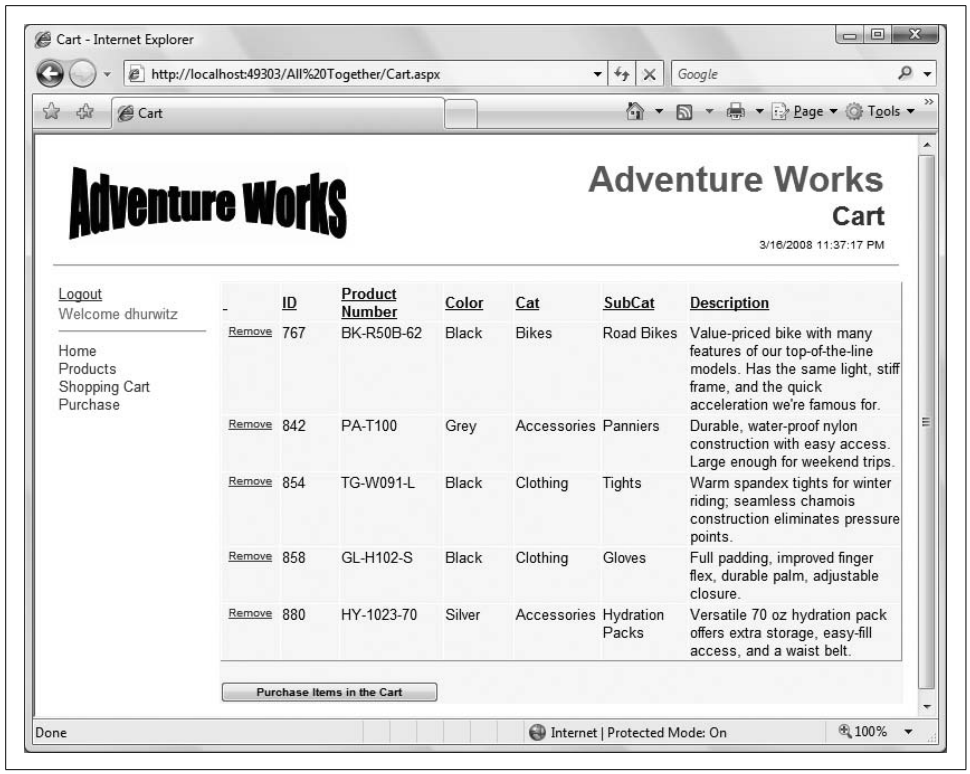

Cart - Internet Explorer
http://localhost:49303/All%20Together/Cart.aspx
Google
Cart
Page
Tools
Adventure Works
Adventure Works
Cart
3/16/2008 11:37:17 PM
Logout
Welcome dhurwitz
Home
Products
Shopping Cart
Purchase
ID
Product Number
Color
Cat
SubCat
Description
Remove 767 BK-R50B-62 Black Bikes Road Bikes Value-priced bike with many features of our top-of-the-line models. Has the same light, stiff frame, and the quick acceleration we're famous for.
Remove 842 PA-T100 Grey Accessories Panniers Durable, water-proof nylon construction with easy access. Large enough for weekend trips.
Remove 854 TG-W091-L Black Clothing Tights Warm spandex tights for winter riding; seamless chamois construction eliminates pressure points.
Remove 858 GL-H102-S Black Clothing Gloves Full padding, improved finger flex, durable palm, adjustable closure.
Remove 880 HY-1023-70 Silver Accessories Hydration Packs Versatile 70 oz hydration pack offers extra storage, easy-fill access, and a waist belt.
Purchase Items in the Cart
Done
Internet | Protected Mode: On
100%

Figure 11-11. *Here's what the cart page looks like after you've added some items to the cart.*

Open *Cart.aspx*. Drag a `SqlDataSource` control onto the content area of the page. Set its ID to `sqlCart`. Configure it similar to the `SqlDataSource` shown previously in Example 11-8. Here is the markup for the control. It looks complex, but really it is a straightforward `SELECT` statement against the `Production.Product` table, with five joins:

```
<asp:SqlDataSource ID="sqlCart" runat="server"
    ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString %>"
    SelectCommand= "select product.ProductID, product.Name, product.ProductNumber,
            product.Color,
            subcat.Name as SubcategoryName, cat.Name as CategoryName,
            description.Description
            from Production.Product product
            join Production.ProductSubcategory subcat on
                product.ProductSubcategoryID = subcat.ProductSubcategoryID
            join Production.ProductCategory cat on
                subcat.ProductCategoryID = cat.ProductCategoryID
            join Production.ProductModel model on
                product.ProductModelID = model.ProductModelID
            join Production.ProductModelProductDescriptionCulture culture on
                model.ProductModelID = culture.ProductModelID
            join Production.ProductDescription description on
                culture.ProductDescriptionID = description.ProductDescriptionID">
</asp:SqlDataSource>
```

Notice the use of *aliases* in the query above. This allows you to control the name of the columns returned. There are two columns called `Name` in this query, from two different tables, which would conflict if you used them both, so you alias `subcat.Name` as `subcategoryName` and `cat.Name` as `categoryName`. Another type of alias is used with the table names in the `from` and `join` clauses, where we alias `Production.Product` as `product` and `Production.ProductSubcategory` as `subcat`.

This query needs a `WHERE` clause. The parameter in the `WHERE` clause needs to come from the `Session` object. ASP.NET actually makes this really easy under some circumstances—but unfortunately not these circumstances, as we will now describe.

You saw previously in Example 11-8 the use of the `<SelectParameters>` element of the `SqlDataSource`, reproduced here, with a parameter based on the value of another control on the page:

```
<SelectParameters>
    <asp:ControlParameter ControlID="gvProducts"
        Name="ProductID" PropertyName="SelectedDataKey.Values['ProductID']" />
</SelectParameters>
```

There are other types of `SelectParameters` controls, including a `SessionParameter`, which comes from a `Session` object. The reason that will not work here is due to a "quirk" of the SQL statement used to construct the query. We'll explain.

The cart is stored in a string as a comma-separated list of `ProductIDs`, which are stored in the database as integers. The query sent to the database has a `where` clause using the `in` keyword, as in:

```
where product.ProductID in (753,845,143) and culture.CultureID = 'en'
```

This `where` clause is created by the `sqlDataSource` control; you don't have to type it in. The two halves of this `where` clause derive from the join specified in the `SelectCommand` of the `SqlDataSource` control.

SQL Server knows that `ProductID` is an integer and is able to parse the contents of the parentheses as a list of integers. However, when you use the `SessionParameter` control, it encloses the contents of the parentheses with quotes, as in:

```
where product.ProductID in ("753,845,143") and culture.CultureID = 'en'
```

The quotes make it a string, and SQL Server cannot parse it as a set of integers. There may be a way to deal with this in SQL, but it is easier, and more instructive, to work around this by writing a handler for the `Selecting` event of the `SqlDataSource` control. This event is raised just before the query is sent to the database, and is a convenient time to modify the query.

Add the code from Example 11-10 to handle this event (as well as events for two controls you will place on the page in just a moment) to *Cart.aspx.vb*. It retrieves the `Session` object and constructs the `where` clause, setting the `CommandText` subproperty of the event argument's `Command` property. Because this event is raised before the

query is executed, changing the `CommandText` of the query allows you to modify the query before it is run; using this technique, you can have the `WHERE` clause refer to a specific `ProductID`.

This example is not as secure as it should be for a production application. At the least, you would want to be careful of passing sensitive information in `Session` this way. When you're constructing SQL statements, all values should be validated and tested to prevent SQL injection attacks, which is well beyond the scope of this book.

Example 11-10. *Cart.aspx.vb event handlers*

```
Protected Sub sqlCart_Selecting(ByVal sender As Object, _
        ByVal e As System.Web.UI.WebControls.SqlDataSourceSelectingEventArgs) _
        Handles sqlCart.Selecting

   Trace.Warn("sqlCart_Selecting") '        to aid in debugging

   Dim strCart As String = String.Empty
   If Session("Cart") IsNot Nothing Then
       strCart = Session("Cart").ToString
        e.Command.CommandText &= " where product.ProductID in (" + _
           strCart + _
           ") and culture.CultureID = 'en' "
   Else
        e.Cancel = True
   End If
End Sub


Protected Sub btnPurchase_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles btnPurchase.Click
    Response.Redirect("Purchase.aspx")
End Sub


Protected Sub gvCart_SelectedIndexChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles gvCart.SelectedIndexChanged
    ' this method is actually hooked to the Remove button & is removing items from the
cart
    Dim strProductID As String = gvCart.SelectedRow.Cells(1).Text
    If Session("Cart") IsNot Nothing Then
        '  remove the selected ProductID from the Session string
        '  Retrieve the session string.
        Dim strCart As String = Session("Cart").ToString( )
        Dim arIDs As String( ) = strCart.Split({",")

        ' iterate through the ID's comprising the string array
        '  rebuild the cart string, leaving out the matching ID
        strCart = String.Empty
```

Example 11-10. *Cart.aspx.vb event handlers (continued)*

```
        For Each str As String In arIDs
            '  use Trim to remove leading and trailing spaces
            If str.Trim( ) <> strProductID.Trim( ) Then
                 strCart += str + ", "
            End If
        Next

        ' remove the trailing space and comma
        If strCart.Length > 1 Then
            strCart = strCart.Trim( )
            strCart = strCart.Substring(0, strCart.Length - 1)
        End If

        ' put it back into Session
        Session("Cart") = strCart

        ' rebind the GridView, which will force the SqlDataSource to requery
        gvCart.DataBind( )
    End If        '  close for test for Session
End Sub
```

In the `sqlCart_Selecting` event handler, we test to see if the `Session` has data. If it does, we add a `where` clause to the SQL command by modifying the `commandText` property of the `Command` object of the `SqlDataSourceSelectingEventArgs`, called `e`. If the `Session` doesn't contain data, the event handler is cancelled by setting the `Cancel` property of `e` to `True`.

Now, drag a `GridView` onto the page, setting its ID to `gvCart`. Configure it similar to the previous `GridView`. Here is the markup for `gvCart`:

```
    <asp:GridView ID="gvCart" runat="server"
            DataSourceID="sqlCart"
            AllowPaging="True" AllowSorting="True" Width="100%"
            AutoGenerateColumns="False"
            HeaderStyle-CssClass="TableColumnHeading"
            RowStyle-CssClass="TableCells">
        <Columns>
            <asp:CommandField ShowSelectButton="True" SelectText="Remove"
                ControlStyle-CssClass="ButtonSelect" ItemStyle-Width="50px"
                    ItemStyle-HorizontalAlign="Center"/>
            <asp:BoundField DataField="ProductID" HeaderText="ID"
                    ItemStyle-Width="50px"/>
            <asp:BoundField DataField="ProductNumber" HeaderText="Product Number"
                    ItemStyle-Width="90px" />
            <asp:BoundField DataField="Color" HeaderText="Color"
                    ItemStyle-Width="60px" />
            <asp:BoundField DataField="CategoryName" HeaderText="Cat"
                    ItemStyle-Width="75px" />
            <asp:BoundField DataField="SubcategoryName" HeaderText="SubCat"
                    ItemStyle-Width="75px" />
            <asp:BoundField DataField="Description" HeaderText="Description" />
        </Columns>
    </asp:GridView>
```

Below the GridView place an HTML <br /> element and an ASP.NET Button control called btnPurchase:

```
<br />
<asp:Button ID="btnPurchase" runat="server" Text="Purchase Items in the Cart"
            CssClass="ButtonText"/>
```

The code to handle the Click event of this button is included in Example 11-10. All this event handler does is redirect readers to the Purchase page, which you'll create shortly.

The Remove button on each row of the GridView was not a normal ASP.NET Button control, but rather a CommandField with its SelectText property set to Remove. Clicking a CommandField in a GridView selects that row of the grid. This is handled with the gvCart_SelectedIndexChanged event handler, included previously in Example 11-10.

Run through the app, logging in and adding some items to the cart. Then switch to the Cart page. You will see something similar to Figure 11-11, shown earlier.

Purchase Page

Clicking on the Purchase button on the Cart page brings you to the Purchase page. This page, as shown in Figure 11-12, is used to gather billing and shipping information from the customer. It has a layout table with a bunch of TextBox controls, a couple of RadioButtonLists, a Buy Now button, and a bunch of associated validation controls.

The first row of the layout table is just a heading. The second row collects the Name. This is a required field, so it has a RequiredFieldValidator.

```
<table border="0" class="TableRowHeading">
    <tr>
        <td colspan="4">
            Billing Information
        </td>
    </tr>
    <tr>
        <td>Name</td>
        <td colspan="3">
            <asp:TextBox ID="txtName" runat="server" Width="250" />
            <asp:RequiredFieldValidator ID="rfName" runat="server"
                ControlToValidate="txtName"
                Display="Dynamic" ErrorMessage="Name is a required field."
                CssClass="ValidationError">*</asp:RequiredFieldValidator></td>
    </tr>
```

All the validation controls on this page will use dynamic display, so room will only be allocated on the page if it is necessary to display the validation text. For this and all the other validation controls, the validation text is simply an asterisk to display

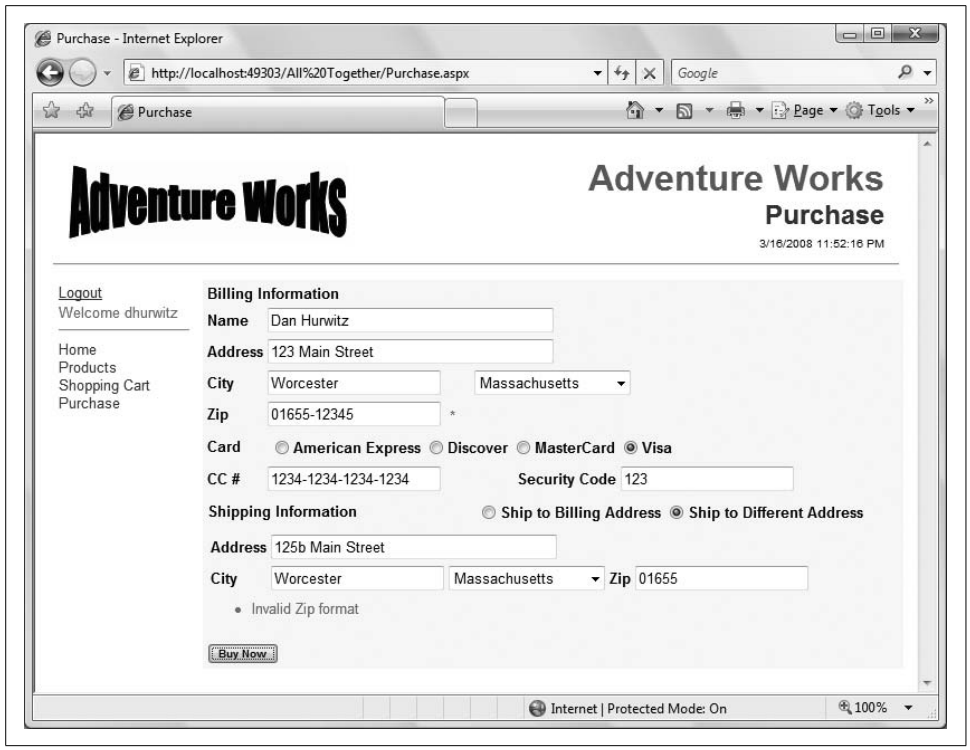


Figure 11-12. *The Purchase page looks like this after you've entered information, including an invalid zip code, and then clicked Buy Now.*

next to the control with invalid input. A `ValidationSummary` control at the bottom of the page will gather all the `ErrorMessages` into a single location.

The next row is the Address, which is very similar to the Name row:

```
<tr>
    <td>Address</td>
    <td colspan="3">
        <asp:TextBox ID="txtAddress" runat="server" Width="250" />
        <asp:RequiredFieldValidator ID="rfAddress" runat="server"
            ControlToValidate="txtAddress"
            Display="Dynamic" ErrorMessage="Address is a required field."
            CssClass="ValidationError">*</asp:RequiredFieldValidator></td>
</tr>
```

Next is a row for both City and State. City is a straightforward `TextBox`, just like Name and Address. However, the State control is a `DropDownList` that is populated from the database. A `SqlDataSource` is used to populate this `DropDownList` and another one further down used as part of the shipping address with a list of state names from the database. It is a very simple query; there are no parameters necessary:

```
<tr>
    <td>City</td>
    <td style="width: 181px">
        <asp:TextBox ID="txtCity" runat="server" />
        <asp:RequiredFieldValidator ID="rfCity" runat="server"
            ControlToValidate="txtCity"
            Display="Dynamic" ErrorMessage="City is a required field."
            CssClass="ValidationError">*</asp:RequiredFieldValidator>
    </td>
    <td colspan="2">
        <asp:SqlDataSource ID="sqlStates" runat="server"
            ConnectionString=
                "<%$ ConnectionStrings:AdventureWorksConnectionString %>"
            SelectCommand="SELECT StateProvinceCode, [Name]
                    FROM Person.StateProvince
                    WHERE CountryRegionCode = 'US' order by [Name]">
        </asp:SqlDataSource>
        <asp:DropDownList ID="ddlStates" runat="server"
            DataSourceID="sqlStates"
            DataTextField="Name" DataValueField="StateProvinceCode" />
    </td>
</tr>
```

The next row gathers the zip code, validated by a `RegularExpressionValidator` to be a valid U.S. zip code, as well as being required. The regular expression requires either five digits or five digits plus four more separated by a dash:

```
<tr>
    <td>Zip</td>
    <td style="width: 181px" colspan="3">
        <asp:TextBox ID="txtZip" runat="server" />
        <asp:RequiredFieldValidator ID="rfZip" runat="server"
            ControlToValidate="txtZip"
            Display="Dynamic" ErrorMessage="Zip is a required field."
            CssClass="ValidationError">*</asp:RequiredFieldValidator>
        <asp:RegularExpressionValidator ID="reZip" runat="server"
            ErrorMessage="Invalid Zip format"
            ControlToValidate="txtZip"
            Display="Dynamic"
            ValidationExpression="^\d{5}$|^\d{5}-\d{4}$"
            CssClass="ValidationError">*</asp:RegularExpressionValidator>
    </td>
</tr>
```

Next is a row to gather credit card information. A `RadioButtonList` allows the user a choice of credit card type, and again validates that the user makes a choice.

Entering credit card numbers in a web site invites fraud. This example makes no pretense of preventing that fraud. A production-quality site would validate the input, including checksums that are built into the credit card number itself. Furthermore, each of the credit card processing companies has its own requirements for what constitutes valid data.

```
<tr>
    <td>Card</td>
    <td colspan="3" >
        <asp:RadioButtonList ID="rblCardType" runat="server"
                RepeatDirection="Horizontal">
            <asp:ListItem Value="am" Text="American Express" />
            <asp:ListItem Value="d" Text="Discover" />
            <asp:ListItem Value="mc" Text="MasterCard" />
            <asp:ListItem Value="v" Text="Visa" />
        </asp:RadioButtonList>
        <asp:RequiredFieldValidator ID="rfCreditCard" runat="server"
            ErrorMessage="Credit Card type is missing."
            ControlToValidate="rblCardType" Display="Dynamic"
            InitialValue=""
            CssClass="ValidationError">*</asp:RequiredFieldValidator>
    </td>
</tr>
```

The next row gathers the credit card number and security code. Both are required and both use a `RegularExpressionValidator` to ensure valid formats:

```
<tr>
    <td>CC #</td>
    <td style="width: 181px">
        <asp:TextBox ID="txtCCNumber" runat="server" />
        <asp:RequiredFieldValidator ID="rfCCNumber" runat="server"
            ControlToValidate="txtCCNumber"
            Display="Dynamic"
            ErrorMessage="Credit Card Number is a required field."
            CssClass="ValidationError">*</asp:RequiredFieldValidator>
        <asp:RegularExpressionValidator ID="reCCNumber" runat="server"
            ErrorMessage="Invalid Credit Card Number"
            ControlToValidate="txtCCNumber"
            Display="Dynamic"
            ValidationExpression=
                "^(\d{4}-){3}\d{4}$|^(\d{4} ){3}\d{4}$|^\d{16}$"
            CssClass="ValidationError">*</asp:RegularExpressionValidator>
    </td>
    <td align="right">Security Code</td>
    <td>
        <asp:TextBox ID="txtSecurityCode" runat="server" />
        <asp:RequiredFieldValidator ID="rfSecurityCode" runat="server"
            ControlToValidate="txtSecurityCode"
            Display="Dynamic"
            ErrorMessage="Security Code is a required field."
            CssClass="ValidationError">*</asp:RequiredFieldValidator>
        <asp:RegularExpressionValidator ID="reSecurityCode" runat="server"
            ErrorMessage="Invalid Security Code"
            ControlToValidate="txtSecurityCode"
            Display="Dynamic"
            ValidationExpression="^\d{3}$"
            CssClass="ValidationError">*</asp:RegularExpressionValidator>
    </td>
</tr>
```

The credit card number formats allowed are any of the following:

```
1234-1234-1234-1234
1234 1234 1234 1234
1234123412341234
```

It drives me batty when web sites require a credit card number with no spaces or dashes. It is so easy to accept those characters and just remove them before submission, and it would greatly reduce input errors. Long numbers are much easier to enter and read when grouped by intervening spaces or dashes.

The security number is simply a three-digit number.

The next row contains a `RadioButtonList` to give the user the choice of shipping to the billing address or a different shipping address. Depending on the selected value of that control, a `Panel` control containing a field for the shipping address is either made visible or not. The code for doing this is contained in an event handler for the `SelectedIndexChanged` event of `rblShippingAddress`, included later in Example 11-11.

```
<tr>
    <td colspan="2">
        Shipping Information
    </td>
    <td colspan="2">
        <asp:RadioButtonList ID="rblShippingAddress" runat="server"
            AutoPostBack="true" RepeatDirection="Horizontal">
            <asp:ListItem Value="billing" Text="Ship to Billing Address"
                    Selected="True" />
            <asp:ListItem Value="different"
                    Text="Ship to Different Address" />
        </asp:RadioButtonList>
    </td>
</tr>
```

`AutoPostBack` is set to `true` so that the page will respond immediately when the user changes the selection. If a different address is required, then a `Panel` contained in the next row is made visible:

```
<tr>
    <td colspan="4">
        <asp:Panel ID="pnlShippingAddress" runat="server" Visible="false" >
            <table border="0">
                <tr>
                    <td>Address</td>
                    <td colspan="3">
                        <asp:TextBox ID="txtShippingAddress" runat="server"
                            Width="250" />
                    </td>
                </tr>
                <tr>
                    <td>City</td>
                    <td>
```

```
                    <asp:TextBox ID="txtShippingCity" runat="server" />
                </td>
                <td>
                    <asp:DropDownList ID="ddlShippingStates"
                        runat="server"
                        DataSourceID="sqlStates"
                        DataTextField="Name"
                        DataValueField="StateProvinceCode" />
                </td>
                <td>Zip</td>
                <td>
                    <asp:TextBox ID="txtShippingZip" runat="server" />
                    <asp:RegularExpressionValidator ID="reShippingZip"
                        runat="server"
                        ErrorMessage="Invalid Zip format"
                        ControlToValidate="txtShippingZip"
                        Display="Dynamic"
                        ValidationExpression="^\d{5}$|^\d{5}-\d{4}$"
                        CssClass="ValidationError">*
                    </asp:RegularExpressionValidator>
                </td>
            </tr>
        </table>
    </asp:Panel>
</td>
</tr>
```

Notice how this `Panel` control itself contains another table for laying out the controls used to gather the shipping address.

Finally, there is a row to contain the `ValidationSummary` control:

```
<tr>
    <td colspan="4">
        <asp:ValidationSummary ID="ValidationSummary1" runat="server"
               CssClass="ValidationError" />
    </td>
</tr>
```

And one more row to contain the `Button` for completing the purchase:

```
    <tr>
        <td colspan="4">
            <asp:Button ID="btnBuy" runat="server" Text="Buy Now"
                   CssClass="ButtonText" />
        </td>
    </tr>
</table>
```

When the Buy Now button is clicked, a real application would process the order, updating the database as necessary. In our simple example, it will stash the order info in `Session` in a Dictionary object, and then call the `Confirm` page for order confirmation. The event handler for the Buy Now button is included in Example 11-11.

Note that in order for the Dictionary object to be properly instantiated, you must include the `Imports` statement. We've placed it at the top of Example 11-11, but in *Purchase.aspx.vb*, the `Imports` statement must appear before the opening `Partial Class Purchase` statement.

Example 11-11. *Purchase.aspx.vb event handlers*

```
Imports System.Collections.Generic

Protected Sub rblShippingAddress_SelectedIndexChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles rblShippingAddress.SelectedIndexChanged
    If rblShippingAddress.SelectedValue = "billing" Then
        pnlShippingAddress.Visible = False
    Else
        pnlShippingAddress.Visible = True
    End If
End Sub

Protected Sub btnBuy_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles btnBuy.Click
    ' stash all the info in a dictionary object going to Session
    Dim dictBuy As Dictionary(Of String, String) = _
        New Dictionary(Of String, String)
    dictBuy.Add("Name", txtName.Text)
    dictBuy.Add("Address", txtAddress.Text)
    dictBuy.Add("City", txtCity.Text)
    dictBuy.Add("State", ddlStates.SelectedValue)
    dictBuy.Add("Zip", txtZip.Text)
    dictBuy.Add("Card", rblCardType.SelectedValue)
    dictBuy.Add("CardNumber", txtCCNumber.Text)
    dictBuy.Add("SecurityCode", txtSecurityCode.Text)

    If rblShippingAddress.SelectedValue = "billing" Then
        dictBuy.Add("ShippingAddress", txtAddress.Text)
        dictBuy.Add("ShippingCity", txtCity.Text)
        dictBuy.Add("ShippingState", ddlStates.SelectedValue)
        dictBuy.Add("ShippingZip", txtZip.Text)
    Else
        dictBuy.Add("ShippingAddress", txtShippingAddress.Text)
        dictBuy.Add("ShippingCity", txtShippingCity.Text)
        dictBuy.Add("ShippingState", ddlShippingStates.SelectedValue)
        dictBuy.Add("ShippingZip", txtShippingZip.Text)
    End If

    Session("BuyerInfo") = dictBuy

    Response.Redirect("Confirm.aspx")
End Sub
```

Just as you did with the Product page, you can spiff up the user experience by wrapping the entire contents of the `Content` control inside an `UpdatePanel`, as shown in the following code snippet:

```
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
        Runat="Server">
    <asp:UpdatePanel id="UpdatePanel1" runat="server">
        <ContentTemplate>

            ... all the content goes here ...
        </ContentTemplate>
    </asp:UpdatePanel>
</asp:Content>
```

Now, run the web site and navigate to the Purchase page, as shown previously in Figure 11-12, after filling in most of the fields along with an invalid zip code.

Confirm Page

The Confirm page in this example does nothing more than retrieve the two Session objects, one containing the cart and one containing the buyer information, and display them on the page, as shown in the finished page in Figure 11-13. The cart is displayed in a `GridView` and the buyer information is displayed in a `ListBox`.

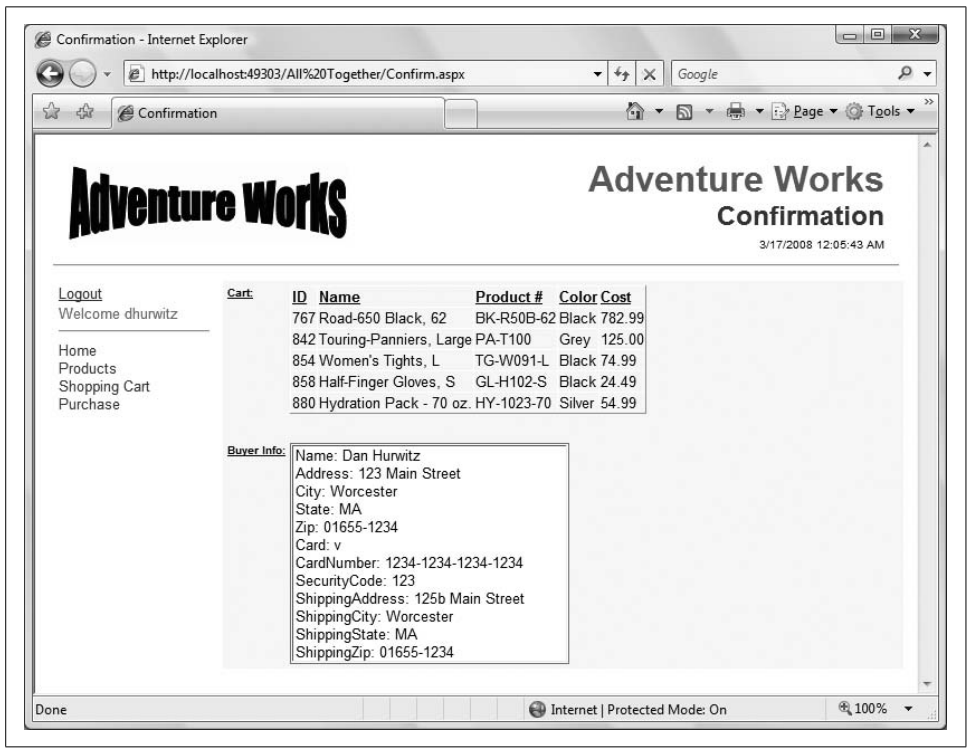


Figure 11-13. *Here's the last page the user will see: the confirmation page. All the information from the Cart and the Purchase page is passed here and displayed again.*

Again, the page contains an HTML table for layout. The first row contains a `GridView` and its associated data source for the cart information:

```
<table>
   <tr>
      <td valign="top" class="ListHeading">
         Cart:
      </td>
      <td valign="top">
         <asp:SqlDataSource ID="sqlCartConfirm" runat="server"
           ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString %>"
            SelectCommand="select ProductID, Name, ProductNumber, Color, ListPrice
                     from Production.Product "></asp:SqlDataSource>
         <asp:GridView ID="gvCart" runat="server"
            DataSourceID="sqlCartConfirm" AllowPaging="True"
            AllowSorting="True" HeaderStyle-CssClass="TableColumnHeading"
            RowStyle-CssClass="TableCells" AutoGenerateColumns="false">
            <Columns>
               <asp:BoundField DataField="ProductID" HeaderText="ID" />
               <asp:BoundField DataField="Name" HeaderText="Name" />
               <asp:BoundField DataField="ProductNumber" HeaderText="Product #" />
               <asp:BoundField DataField="Color" HeaderText="Color" />
               <asp:BoundField DataField="ListPrice" HeaderText="Cost"
                  DataFormatString="{0:F2}" HtmlEncode="false" />
            </Columns>
         </asp:GridView>
      </td>
   </tr>
```

The `SelectCommand` of the `SqlDataSource` is updated with the contents of the Cart Session object in *Confirm.aspx.vb*, exactly as was done for the Cart page, using the `Selecting` event of the `SqlDataSource` control:

```
Protected Sub sqlCartConfirm_Selecting(ByVal sender As Object, _
    ByVal e As System.Web.UI.WebControls.SqlDataSourceSelectingEventArgs) _
    Handles sqlCartConfirm.Selecting

    Trace.Warn("sqlCartConfirm_Selecting") ' aid in debugging

    If Session("Cart") IsNot Nothing Then
        Dim strCart = Session("Cart").ToString
        e.Command.CommandText &= "where ProductID in (" + _
            strCart + ")"
    Else
         e.Cancel = True
    End If

End Sub
```

Back in *Confirm.aspx*, after a spacing row, the next row contains a `ListBox` for the buyer information:

```
    <tr>
        <td colspan="2"> </td>
    </tr>
```

```
        <tr>
            <td valign="top" class="ListHeading">Buyer Info:</td>
            <td valign="top">
                <asp:ListBox ID="lbBuyerInfo" runat="server" Rows="12"
                    Width="250" />
            </td>
        </tr>
    </table>
```

The `ListBox` is populated in `Page_Load` the first time the page is loaded:

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    If User.Identity.IsAuthenticated = False Then
       Response.Redirect("login.aspx")
    End If

    Me.Master.PageSubTitle.Text = "Confirmation"

    If Not IsPostBack Then
        lbBuyerInfo.Items.Clear( )
        If Session("BuyerInfo") IsNot Nothing Then
            Dim dictBuyerInfo As Dictionary(Of String, String) = Nothing
            dictBuyerInfo = CType(Session("BuyerInfo"), _
                Dictionary(Of String, String))
            For Each key As String In dictBuyerInfo.Keys
                lbBuyerInfo.Items.Add(key + ": " + dictBuyerInfo(key))
            Next
        Else
            lbBuyerInfo.Items.Add("There is no buyer info.")
        End If
    End If
End Sub
```

In order for the Dictionary to work in this method, you need to add the following line at the top of the code-behind file to import the proper namespace:

```
Imports System.Collections.Generic
```

The markup for this page is shown later in Example 11-14, and the code-behind is in Example 11-15.

Running the site and navigating through the entire purchase process brings you to the confirmation page shown previously in Figure 11-13.

Custom Error Pages

In case of any errors, you don't want your users to see the ugly generic error page provided by ASP.NET, so you will add some custom error pages, just like you did in Chapter 8. To do so, add the following section to the *web.config* file within the `<system.web>` section:

```
<!-- Valid values of customErrors mode: On, Off, RemoteOnly -->
<customErrors mode="RemoteOnly" defaultRedirect="CustomErrorPage.aspx">
    <error statusCode="400" redirect="CustomErrorPage400.aspx"/>
    <error statusCode="404" redirect="CustomErrorPage404.aspx"/>
    <error statusCode="500" redirect="CustomErrorPage500.aspx"/>
</customErrors>
```

This will provide for specific error pages to cover errors 400, "Bad Request," the ubiquitous 404, "Not Found," and the dreaded 500, "Internal Server Error." It will also specify a generic error page for any error not specifically covered. Setting the mode to `RemoteOnly` means while working on your local machine, you will see the generic error page, with all its helpful information, but remote users will see your custom error pages.

Notice that here the custom error pages have an extension of *.aspx*, rather than the *.htm* used in Chapter 8. This is so they can take advantage of the master pages.

Now you need to actually create those error pages. Add four new pages to the web site called *CustomErrorPage.aspx, CustomErrorPage400.aspx, CustomErrorPage404.aspx*, and *CustomErrorPage500.aspx*. Be sure to check the checkboxes for "Place code in a separate file" and "Select master page."

In the markup file for each of these new pages, add the following `MasterType` directive, after the `Page` directive but before the opening `<asp:Content>` tag:

```
<%@ MasterType TypeName="MasterPage" %>
```

This will allow each page to modify the master page, setting the page subtitle appropriately. To do this, add the following `Page_Load` method to each page:

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    Me.Master.PageSubTitle.Text = "Error"
End Sub
```

Finally, add some content to each page to indicate what the error is and what to do about it. While you are at it, add a `HyperLink` to take the user back to the home page.

The markup for the error pages is too trivial to waste space in the book. They are included with the code download from *www.LibertyAssociates.com*.

Summary

There you have it—a functional web site with user registration, data access, session state, and a consistent look and feel, all coded by you. You can now go out and create sites that you didn't dream were possible just a short time ago. There are more features covered in this book that you could add to this site, such as more complex

validation, user profiles, and LINQ. This example is already fairly extensive, and provides a good start for additional exploration in ASP.NET.

Don't let this be the end of your learning, though. Although you're quite familiar with most of the controls we've discussed, they also have plenty of properties that you can still discover on your own. Experiment with the examples and exercises in this book to see what's possible. The Web is full of ASP.NET resources to continue your education—the AJAX community is adding new extenders all the time, just to pick one example. And, of course, there are other fine books out there, including *Programming ASP.NET* (O'Reilly), to help you learn about the advanced controls.

Source Code Listings

This section contains complete source code listings for the entire site. The style sheet, *StyleSheet.css*, is listed in Example 11-1. The site map file, *Web.sitemap*, is listed in Example 11-7. Examples 11-12 through 11-25 show the markup and code-behind files for each of the pages in the site.

Cart Page

Example 11-12. *Cart.aspx*

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
   CodeFile="Cart.aspx.vb" Inherits="Cart" Title="Cart Page" Trace="false" %>

<%@ MasterType TypeName="MasterPage" %>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
   <asp:SqlDataSource ID="sqlCart" runat="server"
      ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString %>"
      SelectCommand="select product.ProductID, product.Name, product.ProductNumber,
            product.Color,
            subcat.Name as SubcategoryName, cat.Name as CategoryName,
            description.Description
            from Production.Product product
            join Production.ProductSubcategory subcat on
                product.ProductSubcategoryID = subcat.ProductSubcategoryID
            join Production.ProductCategory cat on
                subcat.ProductCategoryID = cat.ProductCategoryID
            join Production.ProductModel model on
                product.ProductModelID = model.ProductModelID
            join Production.ProductModelProductDescriptionCulture culture on
                model.ProductModelID = culture.ProductModelID
            join Production.ProductDescription description on
                culture.ProductDescriptionID = description.ProductDescriptionID">
   </asp:SqlDataSource>
   <asp:GridView ID="gvCart" runat="server" DataSourceID="sqlCart" AllowPaging="True"
      AllowSorting="True" Width="100%" AutoGenerateColumns="False"
      HeaderStyle-CssClass="TableColumnHeading"
      RowStyle-CssClass="TableCells">
```

Example 11-12. *Cart.aspx (continued)*

```
      <Columns>
         <asp:CommandField ShowSelectButton="True" SelectText="Remove"
            ControlStyle-CssClass="ButtonSelect"
            ItemStyle-Width="50px" ItemStyle-HorizontalAlign="Center" />
         <asp:BoundField DataField="ProductID" HeaderText="ID" ItemStyle-Width="50px" />
         <asp:BoundField DataField="ProductNumber" HeaderText="Product Number"
            ItemStyle-Width="90px" />
         <asp:BoundField DataField="Color" HeaderText="Color" ItemStyle-Width="60px" />
         <asp:BoundField DataField="CategoryName" HeaderText="Cat"
            ItemStyle-Width="75px" />
         <asp:BoundField DataField="SubcategoryName" HeaderText="SubCat"
            ItemStyle-Width="75px" />
         <asp:BoundField DataField="Description" HeaderText="Description" />
      </Columns>
   </asp:GridView>
   <br />
   <asp:Button ID="btnPurchase" runat="server" Text="Purchase Items in the Cart"
      CssClass="ButtonText" />
</asp:Content>
```

Example 11-13. *Cart.aspx.vb*

```
Partial Class Cart
    Inherits System.Web.UI.Page

   Protected Sub Page_Load(ByVal sender As Object, _
         ByVal e As System.EventArgs) Handles Me.Load
      If User.Identity.IsAuthenticated = False Then
         Response.Redirect("login.aspx")
      End If

      Me.Master.PageSubTitle.Text = "Cart"

   End Sub

Protected Sub sqlCart_Selecting(ByVal sender As Object, _
        ByVal e As System.Web.UI.WebControls.SqlDataSourceSelectingEventArgs) _
        Handles sqlCart.Selecting

   Trace.Warn("sqlCart_Selecting") '        to aid in debugging

   Dim strCart As String = String.Empty
   If Session("Cart") IsNot Nothing Then
       strCart = Session("Cart").ToString
       e.Command.CommandText &= " where product.ProductID in (" + _
          strCart + _
          ") and culture.CultureID = 'en' "
   Else
        e.Cancel = True
   End If
End Sub
```

Example 11-13. *Cart.aspx.vb (continued)*

```
Protected Sub btnPurchase_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles btnPurchase.Click
    Response.Redirect("Purchase.aspx")
End Sub


Protected Sub gvCart_SelectedIndexChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles gvCart.SelectedIndexChanged
   ' this method is actually hooked to the Remove button &
   '    is removing items from the cart
   Dim strProductID As String = gvCart.SelectedRow.Cells(1).Text
   If Session("Cart") IsNot Nothing Then
     '  remove the selected ProductID from the Session string
     '  Retrieve the session string.
     Dim strCart As String = Session("Cart").ToString( )
     Dim arIDs As String( ) = strCart.Split(New [Char]( ) {","c})

     ' iterate through the ID's comprising the string array
     '  rebuild the cart string, leaving out the matching ID
     strCart = String.Empty
     For Each str As String In arIDs
         '  use Trim to remove leading and trailing spaces
         If str.Trim( ) <> strProductID.Trim( ) Then
              strCart += str + ", "
         End If
     Next

     ' remove the trailing space and comma
     If strCart.Length > 1 Then
         strCart = strCart.Trim( )
         strCart = strCart.Substring(0, strCart.Length - 1)
     End If

     ' put it back into Session
     Session("Cart") = strCart

     ' rebind the GridView, which will force the SqlDataSource to requery
     gvCart.DataBind( )
   End If         '  close for test for Session
End Sub

End Class
```

Confirm Page

Example 11-14. *Confirm.aspx*

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
   CodeFile="Confirm.aspx.vb" Inherits="Confirm" Title="Confirmation Page" Trace="false"
%>

<%@ MasterType TypeName="MasterPage" %>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
   <table>
      <tr>
         <td valign="top" class="ListHeading">
            Cart:
         </td>
         <td valign="top">
            <asp:SqlDataSource ID="sqlCartConfirm" runat="server"
               ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString %>"
               SelectCommand="select ProductID, Name, ProductNumber, Color, ListPrice
                        from Production.Product "></asp:SqlDataSource>
            <asp:GridView ID="gvCart" runat="server" DataSourceID="sqlCartConfirm"
               AllowPaging="True" AllowSorting="True"
               HeaderStyle-CssClass="TableColumnHeading" RowStyle-CssClass="TableCells"
               AutoGenerateColumns="false">
               <Columns>
                  <asp:BoundField DataField="ProductID" HeaderText="ID" />
                  <asp:BoundField DataField="Name" HeaderText="Name" />
                  <asp:BoundField DataField="ProductNumber" HeaderText="Product #" />
                  <asp:BoundField DataField="Color" HeaderText="Color" />
                  <asp:BoundField DataField="ListPrice" HeaderText="Cost"
                     DataFormatString="{0:F2}" HtmlEncode="false" />
               </Columns>
            </asp:GridView>
         </td>
      </tr>
      <tr>
         <td colspan="2">
             
         </td>
      </tr>
      <tr>
         <td valign="top" class="ListHeading">
            Buyer Info:
         </td>
         <td valign="top">
            <asp:ListBox ID="lbBuyerInfo" runat="server" Rows="12" Width="250" />
         </td>
      </tr>
   </table>
</asp:Content>
```

Example 11-15. *Confirm.aspx.vb*

```
Imports System.Collections.Generic      ' neccesary for Dictionary
Partial Class Confirm
    Inherits System.Web.UI.Page

   Protected Sub Page_Load(ByVal sender As Object, _
         ByVal e As System.EventArgs) Handles Me.Load
      If User.Identity.IsAuthenticated = False Then
         Response.Redirect("login.aspx")
      End If

      Me.Master.PageSubTitle.Text = "Confirmation"

      If Not IsPostBack Then
        lbBuyerInfo.Items.Clear( )
        If Session("BuyerInfo") IsNot Nothing Then
            Dim dictBuyerInfo As Dictionary(Of String, String) = Nothing
            dictBuyerInfo = CType(Session("BuyerInfo"),  _
                Dictionary(Of String, String))
            For Each key As String In dictBuyerInfo.Keys
                lbBuyerInfo.Items.Add(key + ": " + dictBuyerInfo(key))
            Next
        Else
            lbBuyerInfo.Items.Add("There is no buyer info.")
        End If
      End If

   End Sub

   Protected Sub sqlCartConfirm_Selecting(ByVal sender As Object, _
       ByVal e As System.Web.UI.WebControls.SqlDataSourceSelectingEventArgs) _
       Handles sqlCartConfirm.Selecting

       Trace.Warn("sqlCartConfirm_Selecting") ' aid in debugging

       If Session("Cart") IsNot Nothing Then
           Dim strCart = Session("Cart").ToString
           e.Command.CommandText &= "where ProductID in (" + _
               strCart + ")"
       Else
            e.Cancel = True
       End If

   End Sub

End Class
```

Home Page

Example 11-16. *Home.aspx*

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
   CodeFile="Home.aspx.vb" Inherits="Home" Title="Home Page" Trace="false" %>

<%@ MasterType TypeName="MasterPage" %>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
   <h2>This Is Your Home Page</h2>
   <div class="TextNormal">
      You can put some stuff about your company here. Perhaps some links. Of course, in
      a real world application, the navigation would probably much more complex. Also,
      the buttons would actually do something, rather than just wave their arms and say
      <span class="TextBold">Look at me!</span>
   </div>
   <asp:Panel ID="pnlEmployee" runat="server" Visible="false">
      <h3>Employee Information</h3>
      <div class="TextNormal">
         This panel should only be visible to users are a members of the <b>Employee</b>
         role. Turning on the visibility of this Panel occurs in the Page_Load event
         handler.
      </div>
   </asp:Panel>
   <asp:Panel ID="pnlManager" runat="server" Visible="false">
      <h3>Manager Information</h3>
      <div class="TextNormal">
         This panel should only be visible to users are a members of the <b>Manager</b>
         role.
         Turning on the visibility of this Panel occurs in the Page_Load event handler.
      </div>
   </asp:Panel>
</asp:Content>
```

Login Page

Example 11-17. *Login.aspx*

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
   CodeFile="Login.aspx.vb" Inherits="Login" Title="Login" Trace="false" %>
<%@ MasterType TypeName="MasterPage" %>

<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
   <asp:Login ID="Login1" runat="server" BackColor="#F7F6F3" BorderColor="#E6E2D8"
      BorderPadding="4" BorderStyle="Solid" BorderWidth="1px"
      DestinationPageUrl="~/Home.aspx"
      Font-Names="Verdana" Font-Size="0.8em" ForeColor="#333333">
      <TextBoxStyle Font-Size="0.8em" />
      <LoginButtonStyle BackColor="#FFFBFF"
         BorderColor="#CCCCCC" BorderStyle="Solid" BorderWidth="1px"
         Font-Names="Verdana" Font-Size="0.8em" ForeColor="#284775" />
      <InstructionTextStyle Font-Italic="True" ForeColor="Black" />
```

Example 11-17. *Login.aspx (continued)*

```
      <TitleTextStyle BackColor="#5D7B9D" Font-Bold="True" Font-Size="0.9em"
         ForeColor="White" />
   </asp:Login>
</asp:Content>
```

Example 11-18. *Login.aspx.vb*

```
Partial Class Login
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, _
            ByVal e As System.EventArgs) Handles Me.Load
        Me.Master.PageSubTitle.Text = "Login"
    End Sub
End Class
```

Master Page

Example 11-19. *MasterPage.master*

```
<%@ Master Language="VB" CodeFile="MasterPage.master.vb"
   Inherits="MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Adventure Works</title>
   <link href="StyleSheet.css" rel="stylesheet" type="text/css" />
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
      <table border="0">
         <tr>
            <td colspan="4">
               <table>
                  <tr>
                     <td width="10px">
                         
                     </td>
                     <td>
                        <asp:ImageButton ID="ibLogo" runat="server"
                           ImageUrl="~/images/AdventureWorksLogo-250x70.gif"
                           AlternateText="AdventureWorks logo"
                           PostBackUrl="~/Home.aspx" />
                     </td>
                     <td width="10px">
                         
                     </td>
                     <td width="500px" align="right">
```

Example 11-19. *MasterPage.master (continued)*

```
                        <span class="PageTitle">Adventure Works</span>
                        <br />
                        <asp:Label ID="lblPageSubTitle" runat="server"
                           CssClass="PageSubTitle" Text="Page SubTitle" />
                        <br />
                        <asp:Label ID="lblTime" runat="server" CssClass="TextXSmall" />
                     </td>
                     <td width="10px">
                         
                     </td>
                  </tr>
                  <tr>
                     <td colspan="5">
                        <hr />
                     </td>
                  </tr>
               </table>
            </td>
         </tr>
         <tr>
            <td width="5px">
                
            </td>
            <td width="150px" valign="top">
               <asp:LoginStatus ID="LoginStatus1" runat="server" CssClass="Hyperlink" />
               <br />
               <asp:LoginView ID="LoginView1" runat="server">
                  <LoggedInTemplate>
                     <span class="WarningRoutine">Welcome</span>
                     <asp:LoginName ID="LoginName1" runat="server"
                        CssClass="WarningRoutine" />
                  </LoggedInTemplate>
                  <AnonymousTemplate>
                     <span class="WarningRoutine">You are not logged in.
                         Please click the login link to log in to this website.</span>
                  </AnonymousTemplate>
               </asp:LoginView>
               <hr />
               <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server"
                   ShowStartingNode="false" />
               <asp:Menu ID="Menu1" runat="server" DataSourceID="SiteMapDataSource1"
                   CssClass="MenuText" />
            </td>
            <td width="5px">
                
            </td>
            <td width="700px" valign="top" bgcolor="yellow">
               <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
               </asp:ContentPlaceHolder>
            </td>
         </tr>
      </table>
```

Example 11-19. *MasterPage.master (continued)*

```
   </div>
   </form>
</body>
</html>
```

Example 11-20. *MasterPage.master.vb*

```
Partial Class MasterPage
    Inherits System.Web.UI.MasterPage
Public Property PageSubTitle( ) As Label
    Get
        Return lblPageSubTitle
    End Get
    Set(ByVal value As Label)
        lblPageSubTitle = value
    End Set
End Property

Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    If Page.User.Identity.IsAuthenticated = False Then
        Menu1.Enabled = False
    End If

    lblTime.Text = DateTime.Now.ToString( )
End Sub
End Class
```

Products Page

Example 11-21. *Products.aspx*

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
   CodeFile="Products.aspx.vb" Inherits="Products" Title="Products Page" %>

<%@ MasterType TypeName="MasterPage" %>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
   <asp:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>
         <asp:SqlDataSource ID="sqlCategories" runat="server"
            ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString %>"
            SelectCommand="select Name, ProductCategoryID from Production.ProductCategory
                           order by Name">
         </asp:SqlDataSource>
         <asp:RadioButtonList ID="rblCategories" runat="server" AutoPostBack="True"
            CssClass="LabelSmall"
            DataSourceID="sqlCategories" DataTextField="Name"
            DataValueField="ProductCategoryID"
            RepeatDirection="Horizontal">
         </asp:RadioButtonList>
         <asp:SqlDataSource ID="sqlProducts" runat="server" ConnectionString=
                     "<%$ ConnectionStrings:AdventureWorksConnectionString %>">
            <SelectParameters>
```

Example 11-21. *Products.aspx (continued)*

```
            <asp:ControlParameter ControlID="rblCategories" Name="ProductCategoryID"
                  PropertyName="SelectedValue" />
         </SelectParameters>
      </asp:SqlDataSource>
      <asp:GridView ID="gvProducts" runat="server" DataKeyNames="ProductID"
         AllowPaging="True" AllowSorting="True" AutoGenerateColumns="False"
         HeaderStyle-CssClass="TableColumnHeading"
         RowStyle-CssClass="TableCells" DataSourceID="sqlProducts">
         <Columns>
            <asp:CommandField ShowSelectButton="True" ItemStyle-Width="50"
               ControlStyle-CssClass="ButtonSelect" />
            <asp:BoundField DataField="ProductID" HeaderText="ID"
               SortExpression="ProductID">
               <ItemStyle Width="50px" />
            </asp:BoundField>
            <asp:BoundField DataField="Name" HeaderText="Name"
               SortExpression="Name">
               <ItemStyle Width="225px" />
            </asp:BoundField>
            <asp:BoundField DataField="ProductNumber" HeaderText="Product Number"
               SortExpression="ProductNumber">
               <ItemStyle Width="90px" />
            </asp:BoundField>
            <asp:BoundField DataField="ListPrice" HeaderText="Cost"
               SortExpression="ListPrice">
               <HeaderStyle CssClass="TableColumnHeadingRight" />
               <ItemStyle CssClass="TableNumberDecimal" Width="60px" />
            </asp:BoundField>
         </Columns>
      </asp:GridView>
      <asp:Panel ID="pnlProduct" runat="server" Visible="false">
         <table width="100%">
            <tr>
               <td valign="top">
                  <asp:Button ID="btnAddToCart" runat="server" Text="Add To Cart"
                     OnClick="btnAddToCart_Click" CssClass="ButtonText" />
                  <div class="ListHeading">Items In Cart</div>
                  <asp:Label ID="lblCart" runat="server" CssClass="TextSmall"
                     Width="90" />
               </td>
               <td valign="top">
                  <asp:SqlDataSource ID="sqlDetailsView" runat="server"
                     ConnectionString="<%$ ConnectionStrings:
                     AdventureWorksConnectionString  %>">
                     <SelectParameters>
                        <asp:ControlParameter ControlID="gvProducts" Name="ProductID"
                           PropertyName="SelectedDataKey.Values['ProductID']" />
                     </SelectParameters>
                  </asp:SqlDataSource>
                  <asp:DetailsView ID="DetailsView1" runat="server"
                     DataSourceID="sqlDetailsView"
                     DataKeyNames="ProductID"
```

Example 11-21. *Products.aspx (continued)*

```
                        AutoGenerateRows="false" CssClass="TableCells" BorderWidth="0"
                        FieldHeaderStyle-CssClass="TableRowHeading"
                        CellSpacing="2" CellPadding="2" Width="500px" Height="50px">
                        <Fields>
                           <asp:BoundField DataField="ProductID" HeaderText="Product ID:"
                              SortExpression="ProductID" />
                           <asp:BoundField DataField="Name" HeaderText="Name:"
                              SortExpression="Name" />
                           <asp:BoundField DataField="ProductNumber"
                              HeaderText="Product #:"
                              SortExpression="ProductNumber" />
                           <asp:BoundField DataField="ListPrice" HeaderText="Cost:"
                              SortExpression="ListPrice"
                              DataFormatString="{0:C}" HtmlEncode="false" />
                           <asp:BoundField DataField="Color" HeaderText="Color:"
                              SortExpression="Color" />
                          <asp:BoundField DataField="CategoryName" HeaderText="Category:"
                              SortExpression="CategoryName" />
                           <asp:BoundField DataField="SubcategoryName"
                              HeaderText="SubCategory:"
                              SortExpression="SubcategoryName" />
                           <asp:BoundField DataField="Description"
                              HeaderText="Description:"
                              SortExpression="Description" />
                        </Fields>
                     </asp:DetailsView>
                  </td>
               </tr>
            </table>
         </asp:Panel>
      </ContentTemplate>
   </asp:UpdatePanel>
</asp:Content>
```

Example 11-22. *Products.aspx.vb*

```
Partial Class Products
    Inherits System.Web.UI.Page

   Protected Sub Page_Load(ByVal sender As Object, _
         ByVal e As System.EventArgs) Handles Me.Load
      If User.Identity.IsAuthenticated = False Then
         Response.Redirect("login.aspx")
      End If

      Me.Master.PageSubTitle.Text = "Products"

      Dim strCommand As String = String.Empty
      strCommand = "select ProductID, Name, ProductNumber, ListPrice from " + _
                     "Production.Product "
      strCommand += "where ProductSubcategoryID in "
      strCommand += "(select ProductSubcategoryID from " + _
                     "Production.ProductSubcategory "
```

Example 11-22. *Products.aspx.vb (continued)*

```
      strCommand += "where ProductCategoryID = "
      strCommand += "@ProductCategoryID)"
      sqlProducts.SelectCommand = strCommand

      strCommand = String.Empty
      strCommand += "select product.*, subcat.ProductSubcategoryID, " + _
                    "subcat.Name as SubcategoryName, "
      strCommand += "cat.ProductCategoryID, cat.Name as CategoryName, "
      strCommand += "model.Name as ModelName, model.CatalogDescription, " + _
                    "model.Instructions, "
      strCommand += "description.Description "
      strCommand += "from Production.Product product "
      strCommand += "join Production.ProductSubcategory subcat on " + _
                    "product.ProductSubcategoryID = subcat.ProductSubcategoryID "
      strCommand += "join Production.ProductCategory cat on subcat.ProductCategoryID = " + _
                    "cat.ProductCategoryID "
      strCommand += "join Production.ProductModel model on product.ProductModelID = " + _
                    "model.ProductModelID "
      strCommand += "join Production.ProductModelProductDescriptionCulture culture on " + _
                    "model.ProductModelID = culture.ProductModelID "
      strCommand += "join Production.ProductDescription description on " + _
                    "culture.ProductDescriptionID = description.ProductDescriptionID "
      strCommand += "where product.ProductID = @ProductID and culture.CultureID = 'en' "
      sqlDetailsView.SelectCommand = strCommand

   End Sub

Protected Sub btnAddToCart_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
    ' the contents of the cart will be saved in a Session object as
    '     a string of comma-delimited values of ProductID's
    Dim strCart As String = String.Empty
    Dim strProductId As String = gvProducts.SelectedDataKey.Value.ToString( )

    If Session("Cart") Is Nothing Then
        strCart = strProductId
    Else
        strCart = Session("Cart").ToString( ) + ", " + strProductId
    End If
    Session("Cart") = strCart
    lblCart.Text = strCart
End Sub

Protected Sub gvProducts_SelectedIndexChanged(ByVal sender As Object, _
   ByVal e As System.EventArgs) Handles gvProducts.SelectedIndexChanged
    pnlProduct.Visible = True
End Sub

Protected Sub gvProducts_RowDataBound(ByVal sender As Object, _
   ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs) _
   Handles gvProducts.RowDataBound
    Dim str As String = String.Empty
```

Example 11-22. *Products.aspx.vb (continued)*

```
    If e.Row.RowType = DataControlRowType.DataRow Then
        Dim cell As TableCell = e.Row.Cells(4) ' ListPrice cell
        Dim nCost As Decimal
        Try
            nCost = CType(cell.Text, Decimal)
            str = nCost.ToString("##,##0.00", Nothing)
        Catch ex As ApplicationException
            str = "n.a."
        Finally
            cell.Text = str
        End Try
    End If
End Sub

Protected Sub rblCategories_SelectedIndexChanged(ByVal sender As Object, _
   ByVal e As System.EventArgs) Handles rblCategories.SelectedIndexChanged
    pnlProduct.Visible = False
End Sub

End Class
```

Purchase Page

Example 11-23. *Purchase.aspx*

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" AutoEventWireup="false"
   CodeFile="Purchase.aspx.vb" Inherits="Purchase" Title="Purchase Page" Trace="false"%>

<%@ MasterType TypeName="MasterPage" %>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
   <asp:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>
         <table border="0" class="TableRowHeading">
            <tr>
               <td colspan="4">
                  Billing Information
               </td>
            </tr>
            <tr>
               <td>
                  Name
               </td>
               <td colspan="3">
                  <asp:TextBox ID="txtName" runat="server" Width="250" />
                  <asp:RequiredFieldValidator ID="rfName" runat="server"
                     ControlToValidate="txtName"
                     Display="Dynamic" ErrorMessage="Name is a required field."
                     CssClass="ValidationError">*</asp:RequiredFieldValidator>
               </td>
            </tr>
            <tr>
```

Example 11-23. *Purchase.aspx (continued)*

```
            <td>
               Address
            </td>
            <td colspan="3">
               <asp:TextBox ID="txtAddress" runat="server" Width="250" />
               <asp:RequiredFieldValidator ID="rfAddress" runat="server"
                  ControlToValidate="txtAddress"
                  Display="Dynamic" ErrorMessage="Address is a required field."
                  CssClass="ValidationError">*</asp:RequiredFieldValidator>
            </td>
         </tr>
         <tr>
            <td>
               City
            </td>
            <td style="width: 181px">
               <asp:TextBox ID="txtCity" runat="server" />
               <asp:RequiredFieldValidator ID="rfCity" runat="server"
                  ControlToValidate="txtCity"
                  Display="Dynamic" ErrorMessage="City is a required field."
                  CssClass="ValidationError">*</asp:RequiredFieldValidator>
            </td>
            <td colspan="2">
               <asp:SqlDataSource ID="sqlStates" runat="server"
                  ConnectionString="<%$ ConnectionStrings:
                  AdventureWorksConnectionString %>"
                  SelectCommand="SELECT StateProvinceCode, [Name]
                              FROM Person.StateProvince
                              WHERE CountryRegionCode = 'US' order by [Name]">
               </asp:SqlDataSource>
               <asp:DropDownList ID="ddlStates" runat="server"
                  DataSourceID="sqlStates" DataTextField="Name"
                  DataValueField="StateProvinceCode" />
            </td>
         </tr>
         <tr>
            <td>
               Zip
            </td>
            <td style="width: 181px" colspan="3">
               <asp:TextBox ID="txtZip" runat="server" />
               <asp:RequiredFieldValidator ID="rfZip" runat="server"
                  ControlToValidate="txtZip"
                  Display="Dynamic" ErrorMessage="Zip is a required field."
                  CssClass="ValidationError">*</asp:RequiredFieldValidator>
               <asp:RegularExpressionValidator ID="reZip" runat="server"
                  ErrorMessage="Invalid Zip format"
                  ControlToValidate="txtZip" Display="Dynamic"
                  ValidationExpression="^\d{5}$|^\d{5}-\d{4}$"
                  CssClass="ValidationError">*</asp:RegularExpressionValidator>
            </td>
         </tr>
```

Example 11-23. *Purchase.aspx (continued)*

```
            <tr>
               <td>
                  Card
               </td>
               <td colspan="3">
                  <asp:RadioButtonList ID="rblCardType" runat="server"
                     RepeatDirection="Horizontal">
                     <asp:ListItem Value="am" Text="American Express" />
                     <asp:ListItem Value="d" Text="Discover" />
                     <asp:ListItem Value="mc" Text="MasterCard" />
                     <asp:ListItem Value="v" Text="Visa" />
                  </asp:RadioButtonList>
                  <asp:RequiredFieldValidator ID="rfCreditCard" runat="server"
                     ErrorMessage="Credit Card type is missing."
                     ControlToValidate="rblCardType" Display="Dynamic"
                        InitialValue=""
                        CssClass="ValidationError">*</asp:RequiredFieldValidator>
               </td>
            </tr>
            <tr>
               <td>
                  CC #
               </td>
               <td style="width: 181px">
                  <asp:TextBox ID="txtCCNumber" runat="server" />
                  <asp:RequiredFieldValidator ID="rfCCNumber" runat="server"
                     ControlToValidate="txtCCNumber"
                     Display="Dynamic"
                     ErrorMessage="Credit Card Number is a required field."
                     CssClass="ValidationError">*</asp:RequiredFieldValidator>
                  <asp:RegularExpressionValidator ID="reCCNumber" runat="server"
                     ErrorMessage="Invalid Credit Card Number"
                     ControlToValidate="txtCCNumber" Display="Dynamic"
                     ValidationExpression="^(\d{4}-){3}\d{4}$|^(\d{4} ){3}\d{4}$|^\d{16}$"
                     CssClass="ValidationError">*</asp:RegularExpressionValidator>
               </td>
               <td align="right">
                  Security Code
               </td>
               <td>
                  <asp:TextBox ID="txtSecurityCode" runat="server" />
                  <asp:RequiredFieldValidator ID="rfSecurityCode" runat="server"
                     ControlToValidate="txtSecurityCode"
                     Display="Dynamic" ErrorMessage="Security Code is a required field."
                     CssClass="ValidationError">*</asp:RequiredFieldValidator>
                  <asp:RegularExpressionValidator ID="reSecurityCode" runat="server"
                     ErrorMessage="Invalid Security Code"
                     ControlToValidate="txtSecurityCode" Display="Dynamic"
                     ValidationExpression="^\d{3}$"
                     CssClass="ValidationError">*</asp:RegularExpressionValidator>
               </td>
            </tr>
```

Example 11-23. *Purchase.aspx (continued)*

```
            <tr>
               <td colspan="2">
                  Shipping Information
               </td>
               <td colspan="2">
                  <asp:RadioButtonList ID="rblShippingAddress" runat="server"
                     AutoPostBack="true" RepeatDirection="Horizontal">
                     <asp:ListItem Value="billing" Text="Ship to Billing Address"
                        Selected="True" />
                     <asp:ListItem Value="different" Text="Ship to Different Address" />
                  </asp:RadioButtonList>
               </td>
            </tr>
            <tr>
               <td colspan="4">
                  <asp:Panel ID="pnlShippingAddress" runat="server" Visible="false">
                     <table border="0">
                        <tr>
                           <td>
                              Address
                           </td>
                           <td colspan="3">
                              <asp:TextBox ID="txtShippingAddress" runat="server"
                                 Width="250" />
                           </td>
                        </tr>
                        <tr>
                           <td>
                              City
                           </td>
                           <td>
                              <asp:TextBox ID="txtShippingCity" runat="server" />
                           </td>
                           <td>
                              <asp:DropDownList ID="ddlShippingStates" runat="server"
                                 DataSourceID="sqlStates"
                                  DataTextField="Name"
                                 DataValueField="StateProvinceCode" />
                           </td>
                           <td>
                              Zip
                           </td>
                           <td>
                              <asp:TextBox ID="txtShippingZip" runat="server" />
                              <asp:RegularExpressionValidator ID="reShippingZip"
                              runat="server"
                                 ErrorMessage="Invalid Zip format"
                                 ControlToValidate="txtShippingZip" Display="Dynamic"
                                 ValidationExpression="^\d{5}$|^\d{5}-\d{4}$"
                                 CssClass="ValidationError">*
                              </asp:RegularExpressionValidator>
                           </td>
```

Example 11-23. *Purchase.aspx (continued)*

```
                        </tr>
                     </table>
                  </asp:Panel>
               </td>
            </tr>
            <tr>
               <td colspan="4">
                  <asp:ValidationSummary ID="ValidationSummary1" runat="server"
                     CssClass="ValidationError" />
               </td>
            </tr>
            <tr>
               <td colspan="4">
                  <asp:Button ID="btnBuy" runat="server" Text="Buy Now"
                     CssClass="ButtonText" />
               </td>
            </tr>
         </table>
      </ContentTemplate>
   </asp:UpdatePanel>
</asp:Content>
```

Example 11-24. *Purchase.aspx.vb*

```
Partial Class Purchase
    Inherits System.Web.UI.Page

   Protected Sub Page_Load(ByVal sender As Object, _
         ByVal e As System.EventArgs) Handles Me.Load
      If User.Identity.IsAuthenticated = False Then
         Response.Redirect("login.aspx")
      End If

      Me.Master.PageSubTitle.Text = "Purchase"

   End Sub

Protected Sub rblShippingAddress_SelectedIndexChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles rblShippingAddress.SelectedIndexChanged
    If rblShippingAddress.SelectedValue = "billing" Then
        pnlShippingAddress.Visible = False
    Else
        pnlShippingAddress.Visible = True
    End If
End Sub

Protected Sub btnBuy_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles btnBuy.Click
    ' stash all the info in a dictionary object going to Session
    Dim dictBuy As Dictionary(Of String, String) = _
```

Example 11-24. *Purchase.aspx.vb (continued)*

```
        New Dictionary(Of String, String)
    dictBuy.Add("Name", txtName.Text)
    dictBuy.Add("Address", txtAddress.Text)
    dictBuy.Add("City", txtCity.Text)
    dictBuy.Add("State", ddlStates.SelectedValue)
    dictBuy.Add("Zip", txtZip.Text)
    dictBuy.Add("Card", rblCardType.SelectedValue)
    dictBuy.Add("CardNumber", txtCCNumber.Text)
    dictBuy.Add("SecurityCode", txtSecurityCode.Text)

    If rblShippingAddress.SelectedValue = "billing" Then
        dictBuy.Add("ShippingAddress", txtAddress.Text)
        dictBuy.Add("ShippingCity", txtCity.Text)
        dictBuy.Add("ShippingState", ddlStates.SelectedValue)
        dictBuy.Add("ShippingZip", txtZip.Text)
    Else
        dictBuy.Add("ShippingAddress", txtShippingAddress.Text)
        dictBuy.Add("ShippingCity", txtShippingCity.Text)
        dictBuy.Add("ShippingState", ddlShippingStates.SelectedValue)
        dictBuy.Add("ShippingZip", txtShippingZip.Text)
    End If

    Session("BuyerInfo") = dictBuy

    Response.Redirect("Confirm.aspx")
End Sub

End Class
```

Web.config

Example 11-25. *web.config*

```
<?xml version="1.0"?>
<configuration>
   <configSections>
      <sectionGroup name="system.web.extensions"
type="System.Web.Configuration.SystemWebExtensionsSectionGroup, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">
         <sectionGroup name="scripting" type="System.Web.Configuration.
ScriptingSectionGroup,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">
            <section name="scriptResourceHandler"
type="System.Web.Configuration.ScriptingScriptResourceHandlerSection, System.Web.
Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
requirePermission="false"
allowDefinition="MachineToApplication"/>
            <sectionGroup name="webServices"
type="System.Web.Configuration.ScriptingWebServicesSectionGroup, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">
```

Example 11-25. *web.config (continued)*

```
            <section name="jsonSerialization"
type="System.Web.Configuration.ScriptingJsonSerializationSection, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
requirePermission="false"
allowDefinition="Everywhere"/>
            <section name="profileService"
type="System.Web.Configuration.ScriptingProfileServiceSection, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
requirePermission="false"
allowDefinition="MachineToApplication"/>
            <section name="authenticationService"
type="System.Web.Configuration.ScriptingAuthenticationServiceSection, System.Web.
Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
requirePermission="false"
allowDefinition="MachineToApplication"/>
            <section name="roleService"
type="System.Web.Configuration.ScriptingRoleServiceSection, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
requirePermission="false"
allowDefinition="MachineToApplication"/>
          </sectionGroup>
        </sectionGroup>
      </sectionGroup>
   </configSections>
   <appSettings/>
   <connectionStrings>
  <add name="AdventureWorksConnectionString" connectionString="Data Source=.\
sqlexpress;Initial
Catalog=AdventureWorks;Integrated Security=True"
   providerName="System.Data.SqlClient" />
 </connectionStrings>
   <system.web>
      <!--
            Set compilation debug="true" to insert debugging
            symbols into the compiled page. Because this
            affects performance, set this value to true only
            during development.

            Visual Basic options:
            Set strict="true" to disallow all data type conversions
            where data loss can occur.
            Set explicit="true" to force declaration of all variables.
        -->
  <roleManager enabled="true" />
  <compilation debug="true" strict="false" explicit="true">
        <assemblies>
           <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089"/>
           <add assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
```

Example 11-25. *web.config (continued)*

```
            <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0,
Culture=neutral,
PublicKeyToken=B77A5C561934E089"/>
            <add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089"/>
         </assemblies>
      </compilation>
      <pages>
         <namespaces>
            <clear/>
            <add namespace="System"/>
            <add namespace="System.Collections"/>
            <add namespace="System.Collections.Generic"/>
            <add namespace="System.Collections.Specialized"/>
            <add namespace="System.Configuration"/>
            <add namespace="System.Text"/>
            <add namespace="System.Text.RegularExpressions"/>
            <add namespace="System.Linq"/>
            <add namespace="System.Xml.Linq"/>
            <add namespace="System.Web"/>
            <add namespace="System.Web.Caching"/>
            <add namespace="System.Web.SessionState"/>
            <add namespace="System.Web.Security"/>
            <add namespace="System.Web.Profile"/>
            <add namespace="System.Web.UI"/>
            <add namespace="System.Web.UI.WebControls"/>
            <add namespace="System.Web.UI.WebControls.WebParts"/>
            <add namespace="System.Web.UI.HtmlControls"/>
         </namespaces>
         <controls>
            <add tagPrefix="asp" namespace="System.Web.UI" assembly="System.Web.
Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
            <add tagPrefix="asp" namespace="System.Web.UI.WebControls"
assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
         </controls>
      </pages>
      <!--
            The <authentication> section enables configuration
            of the security authentication mode used by
            ASP.NET to identify an incoming user.
        -->
      <authentication mode="Forms" />
      <!--
            The <customErrors> section enables configuration
            of what to do if/when an unhandled error occurs
            during the execution of a request. Specifically,
            it enables developers to configure html error pages
            to be displayed in place of a error stack trace.

        <customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.htm">
```

Example 11-25. *web.config (continued)*

```
            <error statusCode="403" redirect="NoAccess.htm" />
            <error statusCode="404" redirect="FileNotFound.htm" />
        </customErrors>
        -->
      <!-- Valid values of customErrors mode: On, Off, RemoteOnly -->
      <customErrors mode="RemoteOnly" defaultRedirect="CustomErrorPage.aspx">
         <error statusCode="400" redirect="CustomErrorPage400.aspx"/>
         <error statusCode="404" redirect="CustomErrorPage404.aspx"/>
         <error statusCode="500" redirect="CustomErrorPage500.aspx"/>
      </customErrors>

      <httpHandlers>
         <remove verb="*" path="*.asmx"/>
         <add verb="*" path="*.asmx" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions, Version=3.5.
0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
         <add verb="*" path="*_AppService.axd" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions, Version=3.5.
0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
         <add verb="GET,HEAD" path="ScriptResource.axd"
type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35" validate="false"/>
      </httpHandlers>
      <httpModules>
         <add name="ScriptModule" type="System.Web.Handlers.ScriptModule, System.Web.
Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
      </httpModules>
   </system.web>
   <system.codedom>
      <compilers>
         <compiler language="c#;cs;csharp" extension=".cs" warningLevel="4"
type="Microsoft.CSharp.CSharpCodeProvider, System, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089">
            <providerOption name="CompilerVersion" value="v3.5"/>
            <providerOption name="WarnAsError" value="false"/>
         </compiler>
        <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb" warningLevel="4"
type="Microsoft.VisualBasic.VBCodeProvider, System, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089">
            <providerOption name="CompilerVersion" value="v3.5"/>
            <providerOption name="OptionInfer" value="true"/>
            <providerOption name="WarnAsError" value="false"/>
         </compiler>
      </compilers>
   </system.codedom>
   <!--
        The system.webServer section is required for running ASP.NET AJAX under Internet
        Information Services 7.0.  It is not necessary for previous version of IIS.
    -->
```

Example 11-25. *web.config (continued)*

```
   <system.webServer>
      <validation validateIntegratedModeConfiguration="false"/>
      <modules>
         <remove name="ScriptModule"/>
         <add name="ScriptModule" preCondition="managedHandler"
type="System.Web.Handlers.ScriptModule, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
      </modules>
      <handlers>
         <remove name="WebServiceHandlerFactory-Integrated"/>
         <remove name="ScriptHandlerFactory"/>
         <remove name="ScriptHandlerFactoryAppServices"/>
         <remove name="ScriptResource"/>
         <add name="ScriptHandlerFactory" verb="*" path="*.asmx"
preCondition="integratedMode"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions, Version=3.5.
0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
         <add name="ScriptHandlerFactoryAppServices" verb="*" path="*_AppService.axd"
preCondition="integratedMode" type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/
>
         <add name="ScriptResource" preCondition="integratedMode" verb="GET,HEAD"
path="ScriptResource.axd" type="System.Web.Handlers.ScriptResourceHandler, System.Web.
Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
      </handlers>
   </system.webServer>
   <runtime>
      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
         <dependentAssembly>
            <assemblyIdentity name="System.Web.Extensions"
publicKeyToken="31bf3856ad364e35"/>
            <bindingRedirect oldVersion="1.0.0.0-1.1.0.0" newVersion="3.5.0.0"/>
         </dependentAssembly>
         <dependentAssembly>
            <assemblyIdentity name="System.Web.Extensions.Design"
publicKeyToken="31bf3856ad364e35"/>
            <bindingRedirect oldVersion="1.0.0.0-1.1.0.0" newVersion="3.5.0.0"/>
         </dependentAssembly>
      </assemblyBinding>
   </runtime>
</configuration>
```

APPENDIX A

Installing the Stuff You'll Need

This book contains lots of practice examples, both ones that you can follow along with, and exercises that you can do yourself. To do them, though, you'll need the right tools: an Integrated Development Environment (IDE), the ASP.NET with AJAX extensions, and a database. Fortunately, everything you need is available in free versions from Microsoft, although you can use the fancier paid versions if you want. In this appendix, we'll walk you through getting all the software you need and installing it. By the end of this appendix, you'll be ready to do all the examples in this book.

What Hardware and Software You'll Need

To build the examples in this book, you'll need a PC running one of the following editions of Windows:

- Windows Vista (any edition)
- Windows XP Home, SP 2
- Windows XP Professional, SP 2
- Windows 2000 Professional, SP 4
- Windows Server 2000, SP 4
- Windows Server 2003, SP 2
- Windows x64 (any edition with the accompanying service pack)

Microsoft recommends that your computer have (at a minimum) a Pentium III 600 MHz, with 1 GHz recommended. Although Microsoft insists you can run with 192 MB of memory, 512 MB is recommended. Many serious programmers find that 1 GB of memory is the minimum for professional work, and the authors have recently stepped up to 4 GB, but this is what we do for a living.

Visual Web Developer, the .NET Framework, the documentation, and SQL Server Express will require nearly two gigabytes of space on your hard drive. A full installation of Visual Studio 2008 will take considerably more.

There are two software environments that will work equally well for this book: Visual Web Developer (VWD) and Visual Studio 2008 (VS). The advantage of VWD is that it is free.

Visual Web Developer

Visual Web Developer (VWD) is a subset of Visual Studio and can be used only to build web sites. That said, it is a full-featured development environment, and it's be all you'll need if all you are doing is developing web sites or web services (and you can't beat the price).

Within the realm of creating web sites, we have found very few limitations of VWD. It will not let you create your own AJAX extender controls, which are mentioned (but not demonstrated) in Chapter 3. Also, it will not allow you to compile and deploy assemblies. Both of these are very advanced topics.

You can beef up VWD by downloading the Reporting Add-in, which consists of the ReportViewer control, an integrated report designer, and report programming interface.

Installing VWD

To get started, head to *http://www.microsoft.com/express/download/*, select the language of your choice in the Visual Web Developer box, and click Download. You'll download *vnssetup.exe* to the chosen location on your local machine. After you've downloaded the installer, double-click it and it, will self-install.

Accept the terms of the license and select which additional features you'd like to install along with the development environment, as shown in Figure A-1. You should select the first two options—the MSDN Express Edition provides documentation that will help you when you get stuck on how to use a control, for example. The SQL Server 2005 Express Edition lets you connect your web applications to databases, and you'll need it for several of the examples in this book, starting in Chapter 4. The Silverlight runtime is a useful tool to have, but we won't be covering Silverlight applications in this book, so that checkbox is optional.

If you already have SQL Server installed on your machine, there is no need to install SQL Server 2005 Express Edition. Either SQL Server 2005 or SQL Server 2005 Express will work fine for the examples in this book.

Microsoft Visual Web Developer 2008 Express Edition Setup
Installation Options
Microsoft
Visual Web Developer 2008
Express Edition
Select the optional product(s) you would like to install:
MSDN Express Library for Visual Studio 2008 (Download Size: 297 MB)
The MSDN Express Library contains a subset of all Visual Studio product documentation. If you do not install the MSDN Express Library, you can still access the product documentation on MSDN Online.
Microsoft SQL Server 2005 Express Edition (x86) (Download Size: 36 MB)
SQL Server Express integrates with Visual Studio to provide basic client-database and server-database capabilities.
Microsoft Silverlight Runtime (Download Size: 1.4 MB)
Microsoft Silverlight is a small browser plug-in that enables rich Web experiences. By installing Silverlight you accept the Silverlight license agreement. Silverlight updates automatically, learn more.
For more information, see the Readme file.
< Previous
Next >
Cancel

Figure A-1. *The Visual Web Developer Installation Options. Be sure to select the first two checkboxes for this book.*

You will then be asked to select the destination folder for installation—accepting the default is fine. You'll also be asked to be sure you are connected to the Internet before proceeding with the installation, as shown in Figure A-2.

If you're using Vista, and you install SQL Server 2005 Express, make sure you check the box for "Add user to the SQL Server Administrator role." If you do not, your regular Vista user account won't have appropriate permissions to modify the databases. You can run both the cmd window and Visual Web Developer as Administrator if you want, but it's easier to simply add your user account to the SQL Server Administrator role.

The installation will proceed, downloading what it needs as it goes. Other than a request to register the software, the installation should pretty much take care of itself.

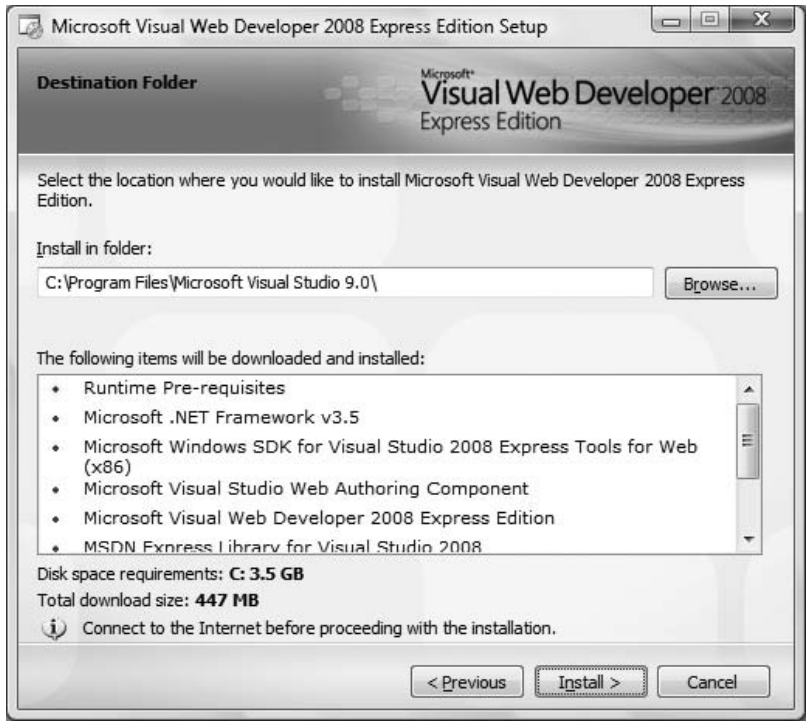

Microsoft Visual Web Developer 2008 Express Edition Setup
Destination Folder
Microsoft
Visual Web Developer 2008
Express Edition
Select the location where you would like to install Microsoft Visual Web Developer 2008 Express Edition.
Install in folder:
C:\Program Files\Microsoft Visual Studio 9.0\
Browse...
The following items will be downloaded and installed:
Runtime Pre-requisites
Microsoft .NET Framework v3.5
Microsoft Windows SDK for Visual Studio 2008 Express Tools for Web (x86)
Microsoft Visual Studio Web Authoring Component
Microsoft Visual Web Developer 2008 Express Edition
MSDN Express Library for Visual Studio 2008
Disk space requirements: C: 3.5 GB
Total download size: 447 MB
Connect to the Internet before proceeding with the installation.
< Previous
Install >
Cancel

Figure A-2. *Before you proceed with the installation, VWD will ask you to select a destination folder; the default works fine. You'll also need to make sure you're connected to the Internet before you click Install.*

After the installation is complete, it is always a good idea to go to the Microsoft web site at *http://msdn2.microsoft.com/en-us/downloads/default.aspx* to get the latest service packs. Once it is fully installed, fire up the program from the Start menu, and the development environment should open, as shown in Figure A-3. You'll see a number of windows here, most of which are empty right now, but you'll be using them a lot once you start creating web pages. The Start Page occupies most of the middle of the screen, with a lot of news from Microsoft that you can browse or ignore, as you see fit. The "Recent Projects" area gives you shortcuts to create a new project or open an existing one; once you've done a few projects, you'll see them listed here for quick access.

The bar on the left is the Toolbox—this is where you'll find the controls you'll be using in your projects. On the upper right is the Solution Explorer, which is empty now, but you'll be able to use it to access any of the files in your project.

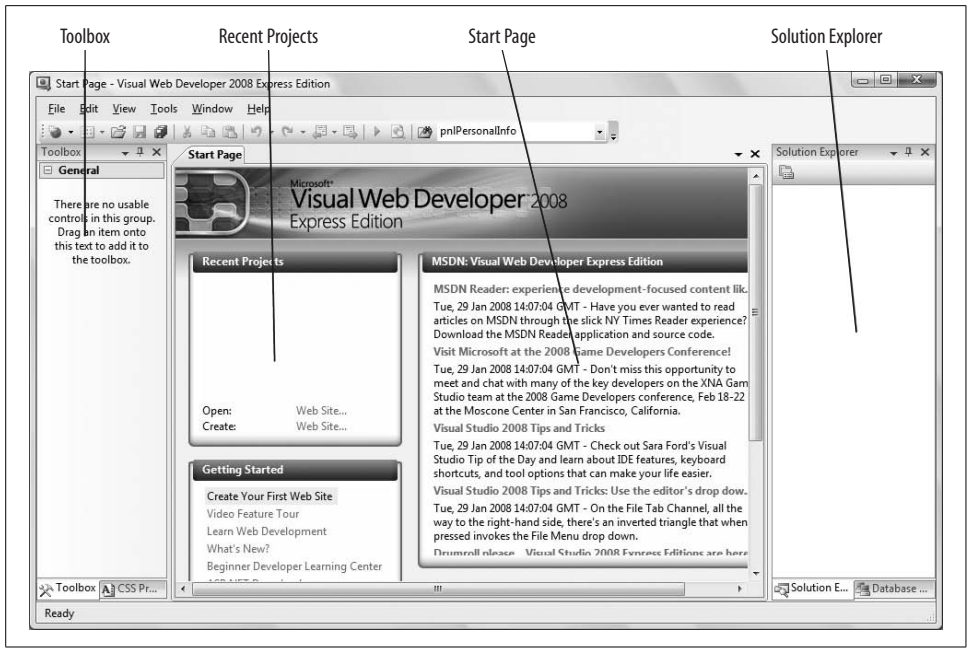

Toolbox
Recent Projects
Start Page
Solution Explorer
Start Page - Visual Web Developer 2008 Express Edition
File Edit View Tools Window Help
pnlPersonalInfo
Toolbox
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
Start Page
Microsoft
Visual Web Developer 2008
Express Edition
Recent Projects
Open: Web Site...
Create: Web Site...
Getting Started
Create Your First Web Site
Video Feature Tour
Learn Web Development
What's New?
Beginner Developer Learning Center
MSDN: Visual Web Developer Express Edition
MSDN Reader: experience development-focused content lik...
Tue, 29 Jan 2008 14:07:04 GMT - Have you ever wanted to read articles on MSDN through the slick NY Times Reader experience? Download the MSDN Reader application and source code.
Visit Microsoft at the 2008 Game Developers Conference!
Tue, 29 Jan 2008 14:07:04 GMT - Don't miss this opportunity to meet and chat with many of the key developers on the XNA Gam Studio team at the 2008 Game Developers conference, Feb 18-22 at the Moscone Center in San Francisco, California.
Visual Studio 2008 Tips and Tricks
Tue, 29 Jan 2008 14:07:04 GMT - Check out Sara Ford's Visual Studio Tip of the Day and learn about IDE features, keyboard shortcuts, and tool options that can make your life easier.
Visual Studio 2008 Tips and Tricks: Use the editor's drop dow...
Tue, 29 Jan 2008 14:07:04 GMT - On the File Tab Channel, all the way to the right-hand side, there's an inverted triangle that when pressed invokes the File Menu drop down.
Drumroll please... Visual Studio 2008 Express Editions are here
Solution Explorer
Toolbox
CSS Pr...
Solution E...
Database ...
Ready

Figure A-3. *The VWD Initial Screen, which you'll see every time you open VWD. There's not much here now, but that'll change shortly.*

Visual Web Developer Service Pack 1 is required for the Browser History example in Chapter 3, and the Dynamic Data example in Chapter 4. All the other examples in this book will work fine without SP1. You can download VWD with SP1 from: *http://www.microsoft.com/express/sp1/*. VWD with SP1 includes the option to install SQL Server Express 2008 instead of SQL Server Express 2005. Because SQL Server Express 2008 was still in beta at the time of this writing, we did not use it to create the examples in this book, and we suggest that you do not install it with VWD with SP1. Instead, install VWD with SP1 by itself, and then install SQL Server Express 2005 separately.

Configuring SQL Server Express

During the VWD installation, SQL Server Express was installed if you checked the appropriate checkbox. To ensure that you can make a connection to your new database, open VWD's Database Explorer window (click View → Database Explorer, or click the Database Explorer tab at the bottom of the Solution Explorer), right-click Data Connections, and choose Add Connection. You will get the dialog box shown in Figure A-4, asking you to choose your data source. Select Microsoft SQL Server, as shown in Figure A-4, and click Continue.

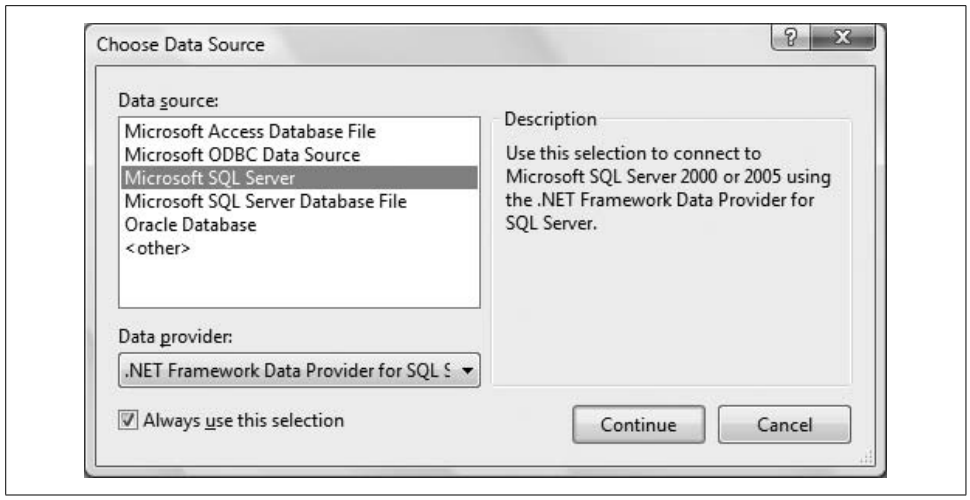

Choose Data Source
Data source:
Microsoft Access Database File
Microsoft ODBC Data Source
Microsoft SQL Server
Microsoft SQL Server Database File
Oracle Database
<other>
Description
Use this selection to connect to Microsoft SQL Server 2000 or 2005 using the .NET Framework Data Provider for SQL Server.
Data provider:
.NET Framework Data Provider for SQL S
Always use this selection
Continue
Cancel

Figure A-4. *To begin the connection to your database, select Microsoft SQL Server as your data source.*

The Add Connection dialog box shown in Figure A-5 will open.

Either type in a server name, use the drop-down menu, or click Refresh to get all the available servers. Typically, however, your SQL Express installation will not appear in the list. If that is the case, just enter **`.\sqlexpress`** (that is, "dot slash" before the word "sqlexpress"). Make sure the radio button "Use Windows Authentication" is chosen, and you should then be able to drop down the list of databases that come with your installation of SQL Express. Select one of those databases to connect to. Once you have selected a database to connect to, click the Test Connection button to verify that the connection is good. Click OK in the Add Connection dialog box.

If you're using SQL Express, you won't be able to use the databases in that list until you install or create a database (or databases) to work with. You need a sample database, such as AdventureWorks, but AdventureWorks doesn't come with SQL Express, so you'll have to install it yourself, which we do in the next section.

Using the AdventureWorks Sample Database

If you're using SQL Express, the sample databases, which we use throughout this book, aren't installed for you. You need to do it manually, and frankly, it's a real pain. The good news is you only need to do it once. If you don't already have the AdventureWorks database installed on your machine, download it from the following:

> *http://www.codeplex.com/MSFTDBProdSamples/Release/ProjectReleases.aspx?ReleaseId=4004*

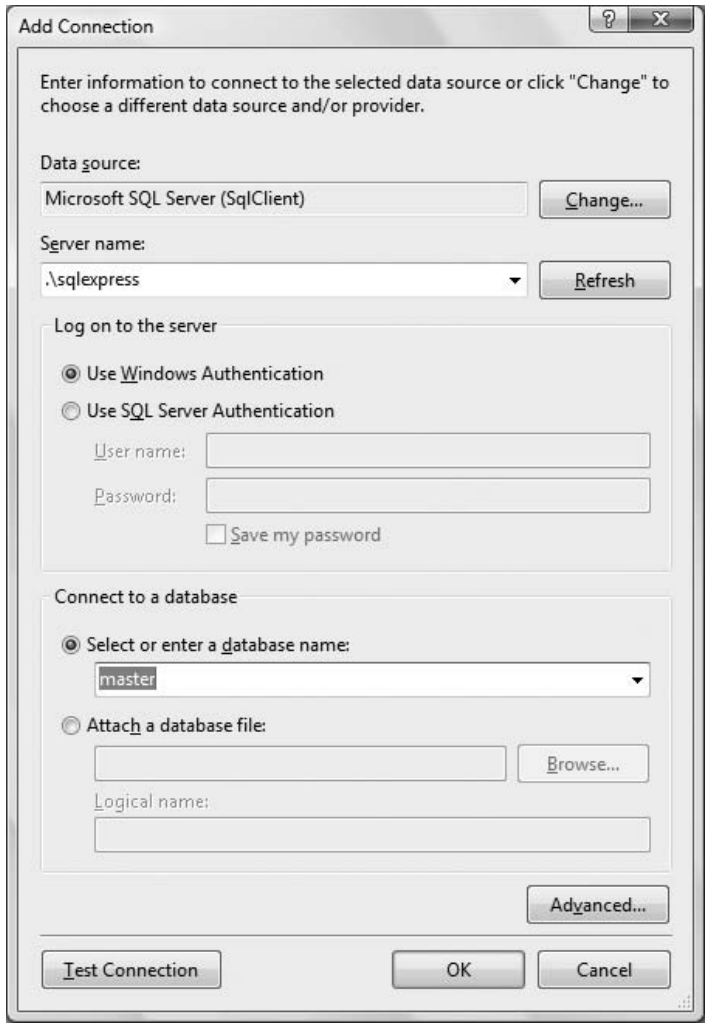


Figure A-5. *VWD Add Connection dialog box. Enter the name of the SQL Server instance as the Server name, select Windows Authentication, and select the database name to connect to.*

There are several different versions of the database you can download, depending on your circumstances and preferences. *AdventureWorksDB.msi* contains the sample database with case-sensitive indexing. *AdventureWorksDBCI.msi* contains a case-insensitive version. Either is fine, but the case-insensitive version is easier to work with. If you are running a 64-bit machine, you should download *AdventureWorksDB_x64.msi*.

Download the appropriate file, and double-click the file to copy the sample database to your machine. You'll be asked to accept the license agreement, and for the file location to copy to—the default is fine.

You next need to attach this sample database to your instance of SQL Express or SQL Server. This is done with a line of SQL script. The exact steps depend on whether you are running SQL Server or SQL Express.

If you are using SQL Server, open a query window in SQL Server Management Studio.

If you are using SQL Express, open a command prompt by clicking Start → Run…, type in `cmd` (in Vista, just click Start and type **`cmd`**), and press Enter. Then, from within that command window, enter the following command to get a query prompt into the database (be careful of your capitalization):

```
sqlcmd -S .\sqlexpress
```

Once you've done that, the cmd window's title will change to SQLCMD, and the prompt will look like this: `1>`.

In either the query window or at the command prompt, enter the following SQL command—all on one line. We've broken it up here to fit on the page:

```
exec sp_attach_db @dbname=N'AdventureWorks', @filename1=N'C:\Program
Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\AdventureWorks_Data.mdf',
@filename2=N'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\Data\AdventureWorks_Log.ldf'
```

In a Management Studio query window, highlight the line of script and press F5 to execute it. In a SQL Express query prompt, type it all in and press Enter. You'll get a new query prompt, `2>`. At this prompt, enter the command **`go`**, and then press Enter again.

If you are using the query prompt, type **`exit`** to quit the program. The database should now be installed and ready to use. If you see messages that indicate that the database is being unpacked, or if you just see the `1>` query prompt, you've done it right. If not, you may have made an error in the SQL command, which is very easy to do.

You can test your installation by attempting to connect to the database, as shown in Figure A-5. If AdventureWorks appears in the list of available databases, you have it right. If not, you may need to try the SQL command again. First check to make sure that the databases are in the correct directory. Go to the default location of `C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\` and see if the two AdventureWorks files are in there. If they're not, you may need to download and install them again, or else use the Windows Find feature to locate the files, and adjust the path in your SQL command accordingly.

Visual Studio 2008

As an alternative to VWD, you can choose to purchase Visual Studio 2008, which comes in a variety of flavors (Standard, Professional, and Team). Briefly, the Standard edition fully supports the creation of ASP.NET web sites. The Professional edition allows you to develop Windows desktop apps, and comes with Crystal Reports and fuller support for XML. Stepping up to the Team edition brings full support for Office development, all languages, 64-bit support, code profiling, static analysis, unit testing, code coverage, project management, and test case management.

Our recommendation: if money is no object or you are part of an enterprise development team, then purchase the top-of-the-line MSDN subscription (Visual Studio Team Suite with MSDN Premium). The approximate retail cost is $11,000 initially and $3,500 to renew. For developers working on smaller projects, either alone or with one or two team members, Visual Studio Professional with MSDN Premium will probably be all you need, at around $3,000 initially and $2,500 to renew. Both of these subscriptions include almost all the software Microsoft sells, plus four support incidents and a free subscription to *MSDN Magazine*. There are many other subscription plans; check them out at *http://msdn2.microsoft.com/en-us/vstudio/aa718657.aspx*.

On the other hand, if you don't feel the need to buy a Ferrari, or even a Corvette, to learn to drive, the free Visual Web Developer and SQL Express will be fine.

Installing Visual Studio 2008

Insert your disk, or click the EXE if you've downloaded the file. On the initial splash screen, click "Install Visual Studio 2008."

Follow the Wizard. Accept the terms of the License Agreement, and when prompted, enter the product key, as shown in Figure A-6.

The next screen in the Wizard, shown in Figure A-7, allows you to select which features to install, as well as the installation location.

The default location is in *c:\Program Files*. In theory, you could install to a different location—say, for example, a hard drive with more space available. This sometimes works, but is often problematic. You will have far fewer problems if you allow the installation to proceed in the default location.

The Default installation requires 4.7 GB of disk space. It installs the most commonly used features, including Visual C#, Visual C++, Visual Basic, Visual Web Developer, the .NET Framework (but without the QuickStart Samples), the Dotfuscator Community Edition, tools for redistributing applications (necessary for deploying

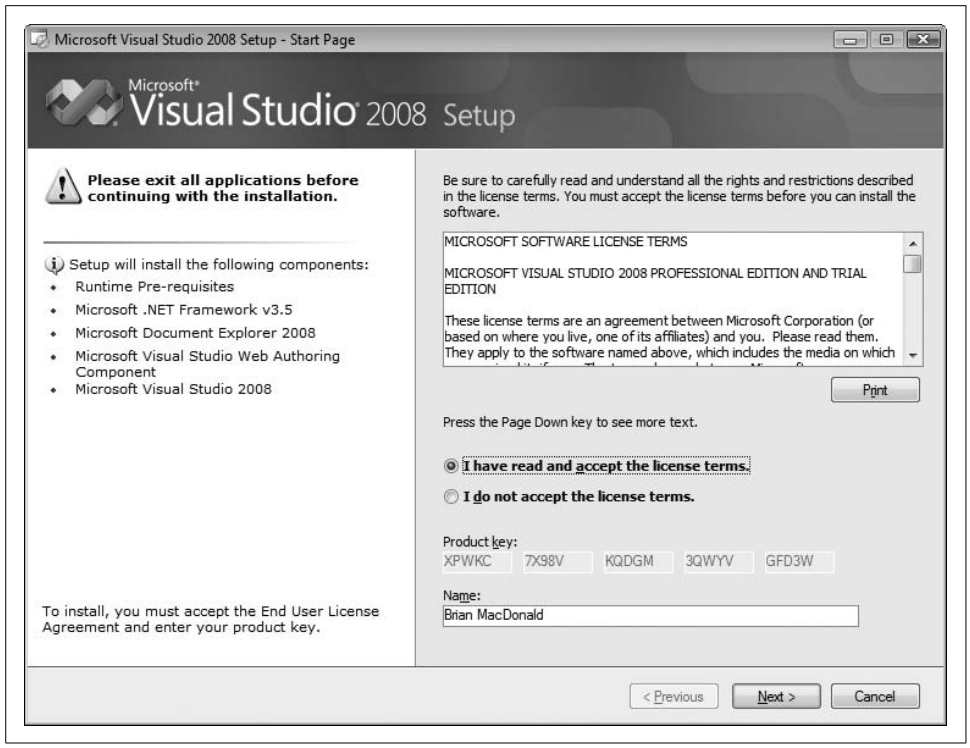

Microsoft Visual Studio 2008 Setup - Start Page
Microsoft
Visual Studio 2008 Setup
Please exit all applications before continuing with the installation.
Setup will install the following components:
Runtime Pre-requisites
Microsoft .NET Framework v3.5
Microsoft Document Explorer 2008
Microsoft Visual Studio Web Authoring Component
Microsoft Visual Studio 2008
To install, you must accept the End User License Agreement and enter your product key.
Be sure to carefully read and understand all the rights and restrictions described in the license terms. You must accept the license terms before you can install the software.
MICROSOFT SOFTWARE LICENSE TERMS
MICROSOFT VISUAL STUDIO 2008 PROFESSIONAL EDITION AND TRIAL EDITION
These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to the software named above, which includes the media on which
Print
Press the Page Down key to see more text.
I have read and accept the license terms.
I do not accept the license terms.
Product key:
XPWKC
7X98V
KQDGM
3QWYV
GFD3W
Name:
Brian MacDonald
< Previous
Next >
Cancel

Figure A-6. *The License Agreement for Visual Studio 2008. Enter your Product key, and click Next to agree to the terms. You can even read the agreement, if you want.*

desktop applications and creating installation packages), Crystal Reports, and SQL Server Express.

The Full installation requires 5.1 GB of disk space. It includes all of the Default installation, plus adds the .NET Framework QuickStart Samples.

The Custom selection allows you to choose what to install, as shown in Figure A-8. You would select this option, for example, if you do not want to install all of the default languages or any of the other features included with the Default installation.

We suggest doing the Custom installation, and then deselecting the language(s) and features that you are quite certain you will never be using. You can always go back and add the missing languages and features. However, it is often very useful to have both C# and VB installed at a minimum, as many examples in articles and other books will be presented in either C# or VB, but not both.

Click the Install button to commence the installation process. The installer will restart your system part-way through, after the Framework is installed, and then continue the installation on its own.

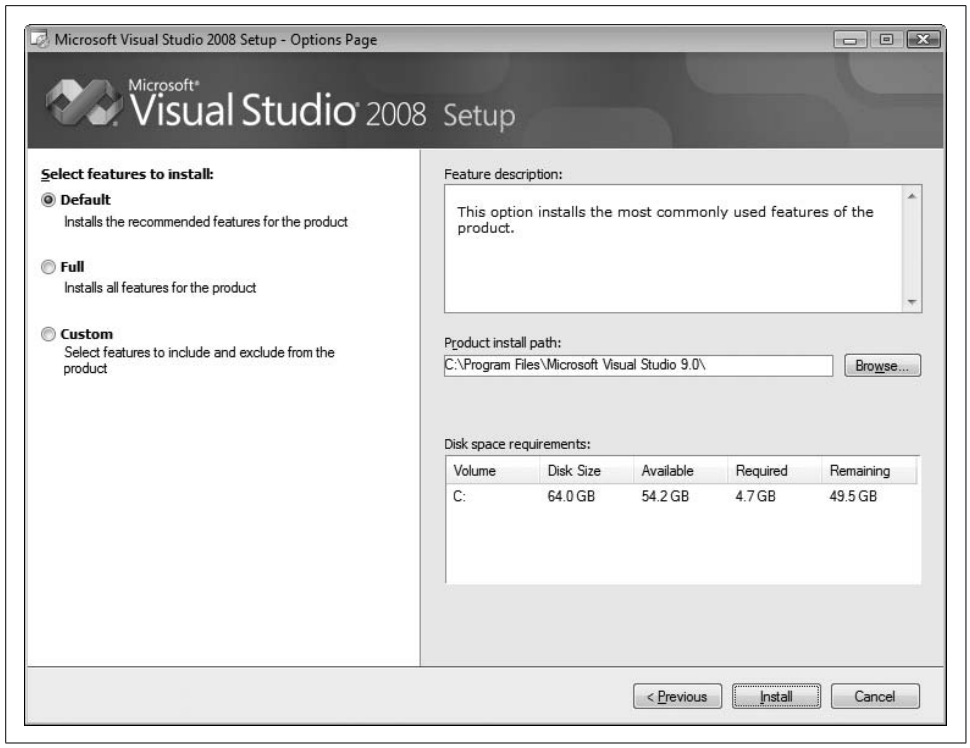


Figure A-7. *This Visual Studio 2008 installation screen lets you customize your installation. Even Default probably has a lot of stuff you don't need, so select Custom and remove the languages that aren't C# or VB.NET.*

After the VS installation completes, you definitely want to install the Product Documentation, which installs the MSDN Library on your machine, if you have the disk space. Do this by clicking the Install Product Documentation link on the splash page. The installation wizard will walk you through the process. Again, you have the choice of a Full (1950 MB of disk space required), Custom, or Minimum (868 MB disk space required) installation. We recommend doing the Full installation, as long as you have the disk space.

Visual Studio 2008 Service Pack 1 is required for the Browser History example in Chapter 3, and the Dynamic Data example in Chapter 4. All the other examples in this book will work fine without SP1. You can download VS 2008 with SP1 from *http://msdn.microsoft.com/en-us/vstudio/default.aspx*.

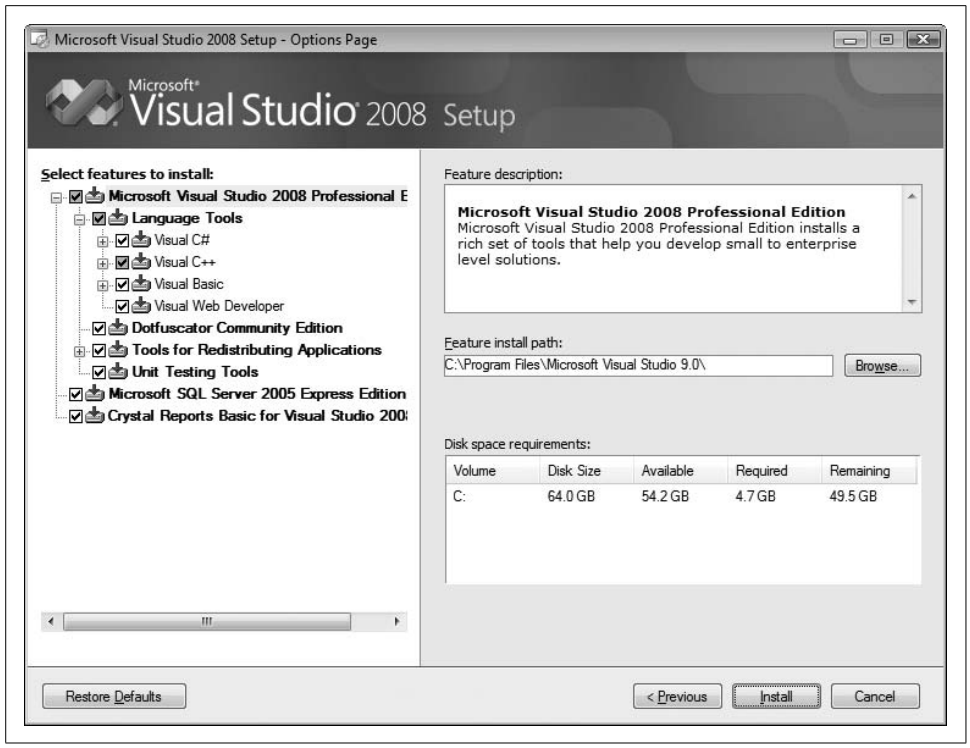


Figure A-8. *The Custom installation option lets you choose which components you want to install. You might not need to install Visual C++ or Crystal Reports, for example.*

ASP.NET AJAX Control Toolkit

The next step is to download and install the AJAX Control Toolkit from Microsoft, available at no charge at *http://www.asp.net/ajax/ajaxcontroltoolkit/samples/*. The AJAX Control Toolkit is an ever-expanding shared-source collection of samples and components, provided by Microsoft and the ASP.NET community, that make it easy to add client-side functionality to your ASP.NET web site. Our experience is that the Toolkit is incredibly valuable, as demonstrated in Chapter 3. Don't be put off by the fact that it is open source; this is solid code and truly worthwhile. However, it is not directly supported by Microsoft, other than in the ASP.NET online forums.

1. Go to *http://www.codeplex.com/AtlasControlToolkit/Release/ProjectReleases.aspx*, and click the Releases tab to find the downloads. There are two versions: *AjaxControlToolkit-Framework3.5.zip* and *AjaxControlToolkit-Framework3.5-NoSource.zip*. The first contains the source code for the components it contains, while the second contains no source code. For this book, all you need is the NoSource file.

This file is updated frequently, so you may want to check back periodically for bug fixes and new features. Be aware that installing a newer version may break stuff that currently works. Usually this is worth the tradeoff.

2. Unzip the AJAX Control Toolkit archive folder into a convenient folder on your computer.
3. Create a new web site by opening Visual Studio 2008 or Visual Web Developer (VS/VWD) and clicking File → New Web Site. Pick "ASP.NET Web Site" under "Visual Studio installed templates." (Yes, we know this is a strange way to install software, but trust us, it is necessary.) You can use the default name for this new web site because it is not necessary to actually save it.
4. After the web site opens in the IDE, the Toolbox should be visible on the left side of the screen. (If it is not visible, click the View → Toolbox menu item.)
5. The Toolbox comprises a number of groups. Each group's name is in boldface, and each group may have a number of items in it; the items are revealed when you click the + preceding the group's name. The first item in the Standard group, for example, is Pointer. What you will do next is add a new group (tab) to the Toolbox with the name "AJAX Control Toolkit," and then populate that group. (Microsoft has made this somewhat confusing to explain because they refer to these groups as both groups and tabs.)
   a. First, check whether the Toolbox already has a group named AJAX Control Toolkit. If it does, skip this step.
   b. Right-click any item in the Toolbox and select "Add Tab."
   c. A text box will open on the Toolbox, above the General tab.
   d. Type the name of the new tab into this text box: "AJAX Control Toolkit"; then press Enter.
6. Right-click the new Toolbox tab and select "Choose Items" from the pop-up menu.
7. When the "Choose Toolbox Items" dialog appears, click the "Browse" button. Navigate to the folder where you unzipped the ASP.NET AJAX Control Toolkit package. You will find a folder named *SampleWebSite*, and under that another folder named *Bin*. Inside that folder, select *AJAXControlToolkit.dll* and click Open. Click OK to close the Choose Toolbox Items dialog.
8. Close the web site. There is no need to save anything.
9. The controls from the AJAX Control Toolkit are now available to use in any of your web sites.

APPENDIX B

Copying a Web Site

You'll often find it necessary or convenient to make a copy of an existing web site—that is, to make a new web site that is the same as the original except for a different name. We do this frequently in this book when building up examples, layering functionality on to a previous example. In the real world, you might want to make a copy of a web site so you can experiment without breaking something that works. We often copy a web site at various stages of development to have an easy snapshot to refer to without having to go to the bother of restoring from backup.

Before looking at the different ways to copy a web site, we'll explain a bit about what actually constitutes a web site. However, if all you want is the cookbook recipe—the set of steps you need to follow to copy a web site—you can skip the following discussion and move on to the next two sections, "Copying the Web Site Without Using the IDE" and "Copying the Web Site with the IDE."

Virtual Directories

Physically, what comprises a web site? A folder on the hard drive of the web server. If the server in question, such as Microsoft IIS (Internet Information Services), is operating outside the bounds of Visual Studio, Visual Web Developer, or some other development tool, then the folder containing the web site must be designated as a *virtual directory*—that is, a directory that is mapped to a web URL by the web server. When a user enters that URL into a browser, the request is passed to the web server and the server looks to the contents of the virtual directory to satisfy the request. How you designate the virtual directory depends on whether you're operating from inside or outside the IDE.

Installing IIS on any of the Home editions of Windows XP, or the Home editions of Windows Vista, is a difficult task. However, not having IIS won't prevent you from copying a web site, or from doing any of the examples in this book.

Outside the IDE

You can map any physical directory on the web server to a virtual directory in IIS. There are several ways to do so. Perhaps the easiest is to click the Start button, right-click My Computer, and select Manage to bring up the Computer Management window. Drill down through Services and Applications, Internet Information Services, and Web Sites to Default Web Site. Alternatively, go to Control Panel → Administrative Tools → Internet Information Services. In Vista, if you start from Computer Management, the Internet Information Services Manager will appear within the Computer Management window, which is tricky to read. It's better to take open IIS Manager from Control Panel in Vista, where you'll see something that looks like Figure B-1.

Figure B-1. *Internet Information Services Manager window in Windows Vista.*

You can see in Figure B-1 that there are already two virtual directories in the Default Web Site, called `FormsBasedSecurityIIS` and `FormsBasedSecurityWAT`. (These correspond to two of the examples in this book from Chapter 9.)

By default, the physical directory corresponding to the Default Web Site virtual directory is located at *c:\inetpub\wwwroot*. You can see this in Windows XP by right-clicking Default Web Site, selecting Properties, and then clicking the Home Directory tab. In Vista, right-click Default Web Site, and select Advanced Settings, and you'll see the physical path in the Advanced Settings dialog, as shown in Figure B-2.

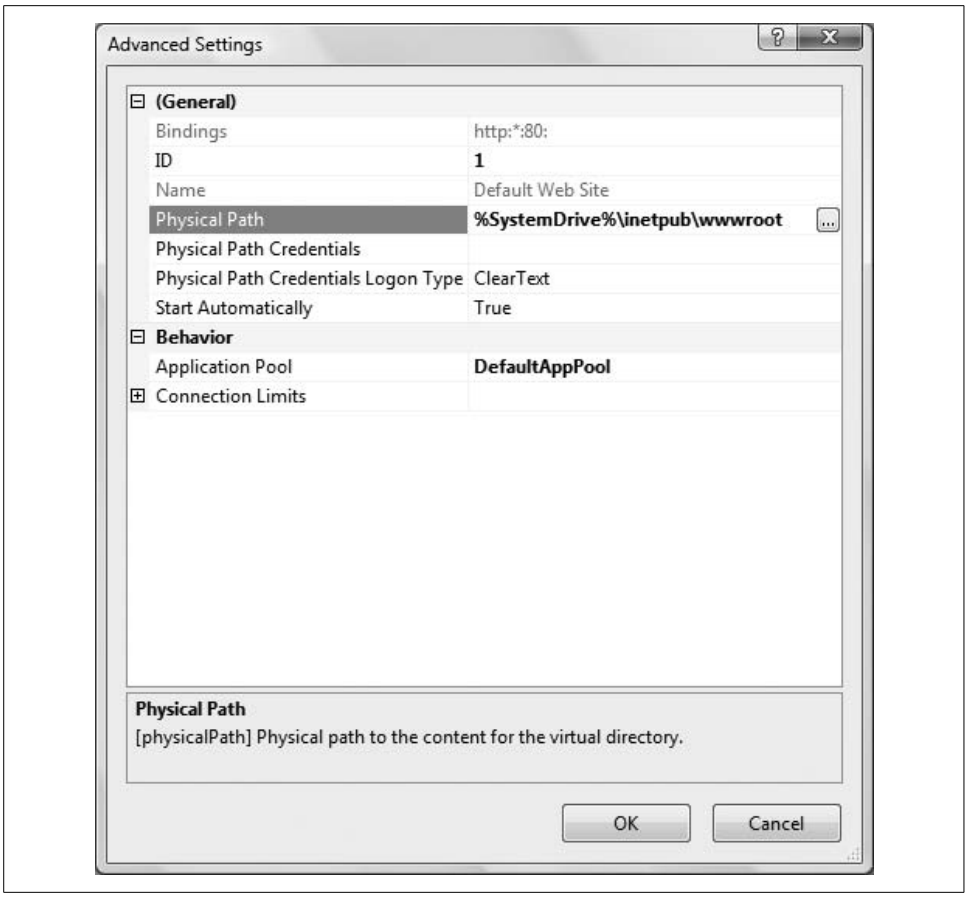


Figure B-2. *You can change the physical directory of your default web site with the Default Web Site Properties dialog.*

If there were a web page called *Welcome.aspx* in the default web site, and the domain name `MyDomain.com` was registered to the IP address of the server, then the following URL in a browser would bring up the page:

```
www.MyDomain.com/Welcome.aspx
```

Opening a browser locally on the server, you would use the following equivalent URL:

```
localhost/Welcome.aspx
```

where `localhost` always refers to the Default Web Site on the local server.

In IIS, certain web page names will be the default for each web site. In other words, you do not need to include them as part of the URL. If you enter a URL without a page name, it will automatically look for one of the default names.

You can see the default page names by right-clicking the virtual directory, selecting Properties, and clicking the Documents tab.

Inside the IDE

One of the big advantages to using Visual Studio or Visual Web Developer is that you do not need to use IIS to serve your pages while in development, and you do not need to create a virtual directory. Instead, the IDE provides its own web server and temporarily creates any necessary virtual directories.

You create a web site in the IDE either by clicking Create Web Site on the Start Page, or by clicking the File → New Web Site menu item. In either case, you get the New Web Site dialog box shown in Figure B-3.

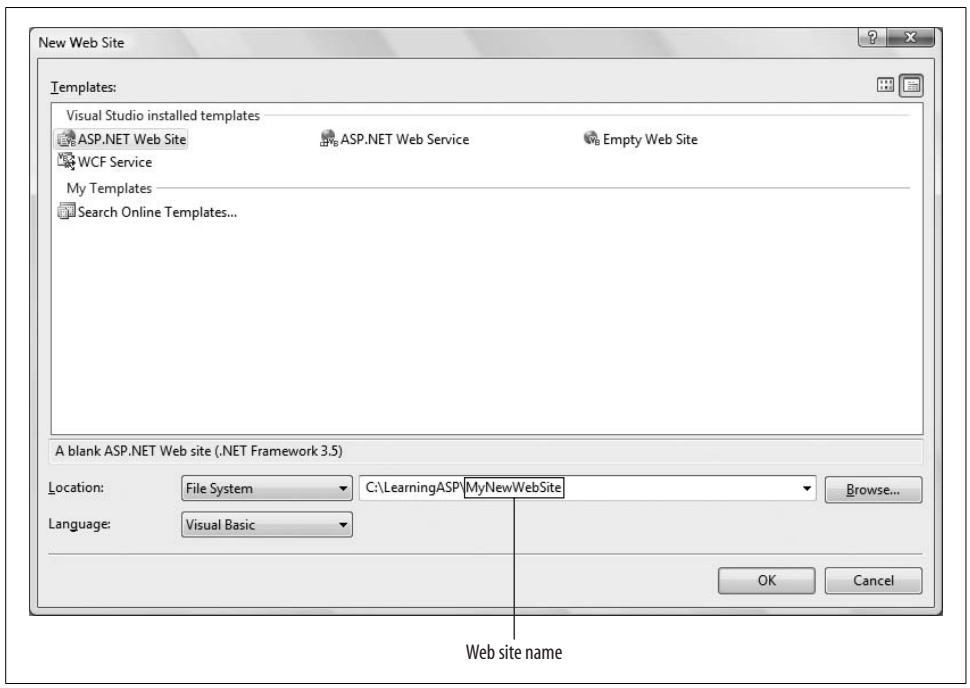


Figure B-3. *New Web Site dialog box with web site path entered and the web site name indicated.*

The highlighted portion of the path in Figure B-3 has a dual meaning. Physically, it is the folder that contains the web site. (If you want to make this web site accessible via IIS, this is the folder that you would make into a virtual directory.) It also is the name of the web site for the IDE.

In other words, when you click Open Web Site on the Start page or the File → Open Web Site menu item, you navigate to this folder to open the web site. The folder contains all the files comprising the web site.

The default location for a new web site will be the location used the last time you created a new web site. If you change the location and name in Figure B-3 to something like *c:\WebSites\OrderEntrySystem*, then the next time you create a new web site, the default location and name will be *c:\WebSites\WebSite2* (assuming there is already a web site named *WebSite1* in that folder).

There is one more piece of this puzzle that will be helpful to know. The IDE keeps track of files that comprise the web site, the default language, and build information. This information is saved in a text file, referred to as the *solution file*, with the same name as the web site (MyNewWebSite in Figure B-3) and an extension of *.sln*. You can open the web site in the IDE by double-clicking the solution file in Windows Explorer. The solution file is used by the IDE for certain housekeeping tasks, but it's not essential, and it is not part of a production web site.

By default, in Windows XP, this solution file is created in a folder with the same name as the web site in *c:\Documents and Settings\<username>\My Documents\Visual Studio 2008\Projects* (or *C:\Users\<username>\Documents\Visual Studio 2008\Projects* in Vista) where <username> is replaced with your user name. You can change this default location by opening the IDE and clicking Tools → Options… → Projects and Solutions → General and changing the Visual Studio projects location, as shown in Figure B-4.

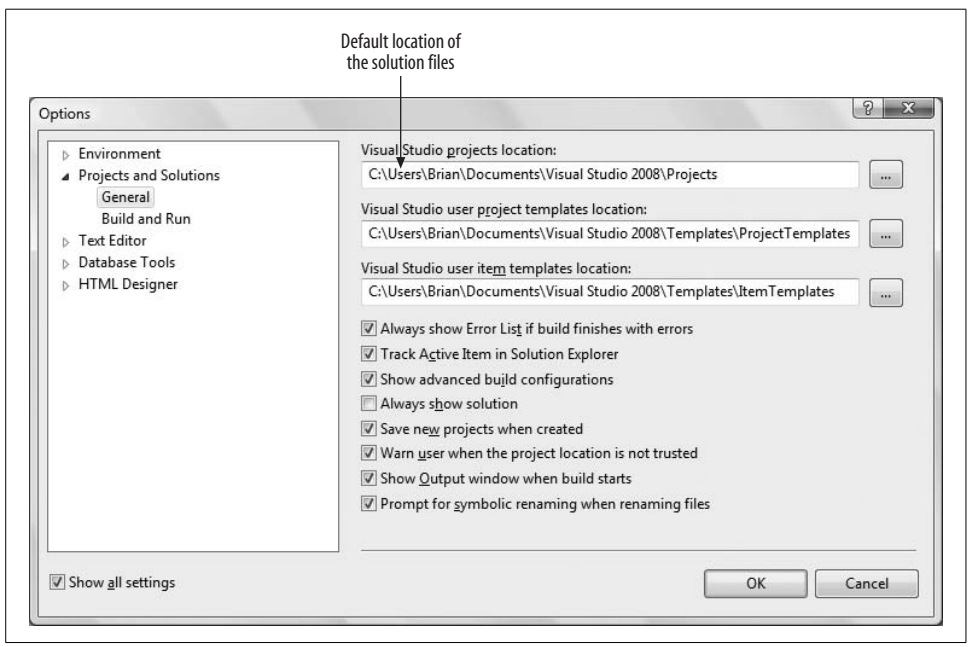


Figure B-4. *Changing the location of the solution files in the Options dialog box.*

Be sure to check the Show all settings checkbox in the lower-left corner of the Options dialog box; otherwise, you will not see many of the options, including the default locations shown in Figure B-4.

There is one more file created with the same name as the solution file and an extension of *.suo*. This file contains developer-specific information relating to the web site, such as which files are displayed in the editing surface, which page is the start page for the web site, breakpoints, and so on. If this file is deleted, or otherwise missing, a new one is automatically created the next time the web site is opened in the IDE.

Now that you know how the files that make up a web site are organized, you will better understand how to copy a web site. There are at least two different ways to do this: inside the IDE and outside the IDE.

Copying the Web Site Without Using the IDE

To copy a web site without using the IDE, simply copy the web site folder to another location and name, using Windows Explorer. The new copy can be in the same parent folder as the original, or in a totally different location. For example, suppose the original web site is called OrderEntrySystem and is located in the following folder:

c:\WebSites\OrderEntrySystem

You want to copy it to a new name, say OrderEntrySystemTest. Copy the original folder to the following:

c:\WebSites\OrderEntrySystemTest

To work with this new web site, open the IDE, click Open Web Site, and navigate to this new folder. That's it.

The IDE will automatically create the necessary solution file and put it in the default location.

Copying the Web Site with the IDE

You can also copy a web site from within the IDE. The advantage of doing it inside the IDE instead of in Windows Explorer is that you have a lot more flexibility. Not only can you copy the site to your local file system, but you can also simultaneously create an IIS virtual directory or copy it to an FTP or remote web site over the Internet.

In this example, we will copy the AdventureWorks web site to another web site, called AdventureWorksRevisited, in the same parent folder.

There are two equivalent ways to begin the process of copying a web site from within the IDE. One is to click the Website → Copy Web Site… menu item. The other is to click the Copy Web Site icon at the top of the Solution Explorer (see Figure B-5).

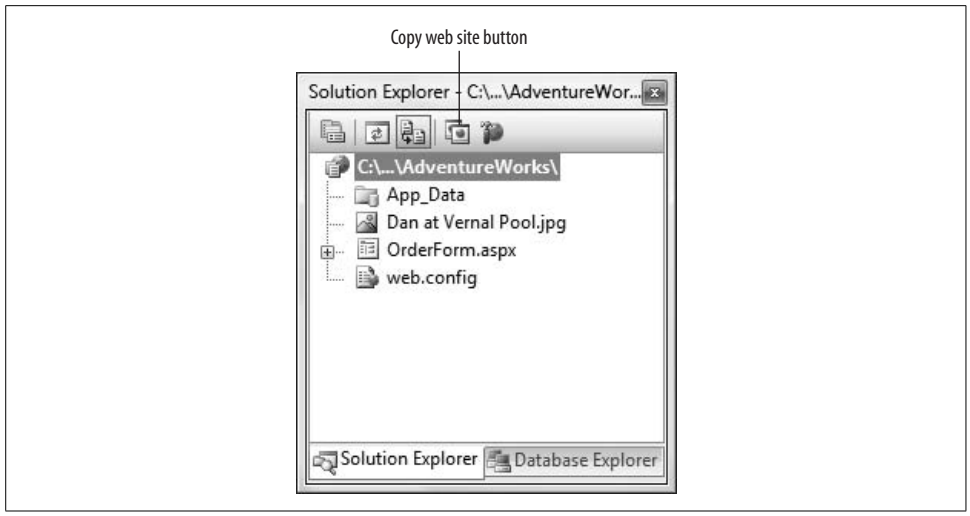


Figure B-5. *The Copy Web Site icon at the top of the Solution Explorer is a quick way to start the copying process.*

Either way you do it, the Copy Web Site window will open in the middle window of the IDE, as shown in Figure B-6.

Click the Connect button, indicated in Figure B-6, to bring up the Open Web Site dialog box shown in Figure B-7. This dialog is used to select the target destination of the copied web site.

The screenshots in these figures were taken after most of the example web sites in this book were already created. Obviously, the folders you see in your file system will be different.

Notice the four icons down the left side of the dialog box. In Figure B-7, the File System icon is selected and the local file system is shown with the source web site initially highlighted.

Clicking any of the other three location icons will replace the File System browser in Figure B-7 with the appropriate means of specifying the location. For instance, if you click FTP site, you will be offered fields for the name of the FTP server and login credentials.

Because you want to copy the target web site to another location within the same parent folder, click that parent folder (Chapter 2, in this case), and then click the Create New Folder icon, indicated in Figure B-8. This will create a new folder under LearningASP\Chapter 2 called WebSite, indicated in Figure B-8, ready to be renamed to something more meaningful.

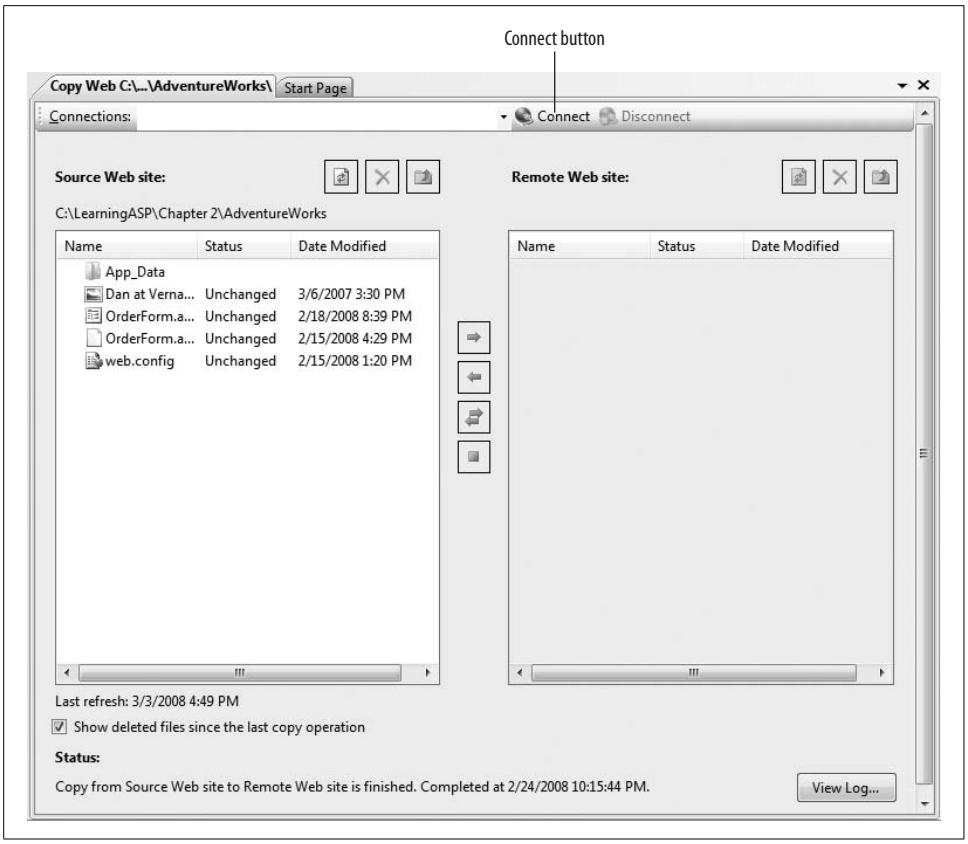


Figure B-6. *The Copy Web Site window looks like this when you first open it. You use the Connect button to locate the destination folder.*

A very common mistake to make here is to forget to select the parent directory before clicking the Create New Folder icon. Because the dialog initially opens with the source folder selected, this mistake will create the new web site folder as a subdirectory of the source folder, which is almost certainly *not* what you intended.

Replace `WebSite` with the target name—in this case, `AdventureWorksRevisited`—and press tab to move off the menu tree. The new folder will be created and the full folder name inserted into the text box.

Click the Open button to select the folder you just created as the target.

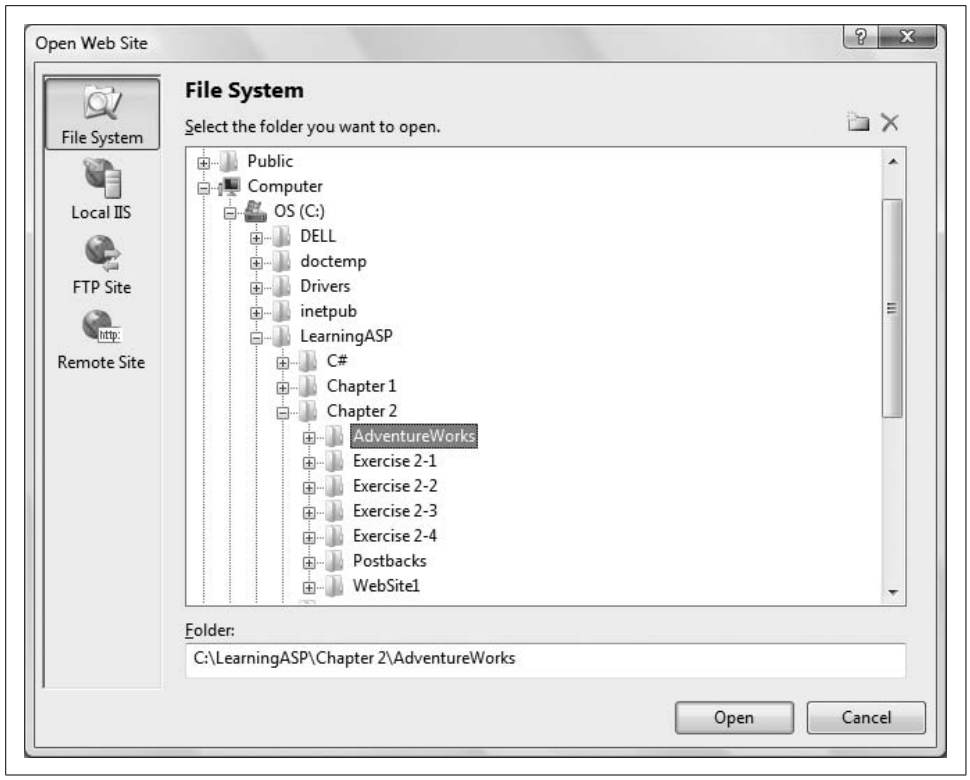


Figure B-7. *The Connect button brings up the Open Web Site dialog box, where you choose the destination for the copy.*

You will be brought back to the Copy Web Site window, similar to that shown back in Figure B-6, except now the target location will be indicated as the Remote Web Site, as shown in Figure B-9. Note that although the target folder here is designated as the "remote" folder, in this case, it's not really remote, it's just another folder on the same computer.

Now you can select the items in the grid on the Source side of the window that you want to copy, which is typically all of them. Click the first item in the list, then hold down Shift and click the last one. When items are selected, the buttons in the middle between the two grids become active. Click the right-pointing arrow to copy over the selected files and folders. The finished result will look something like that shown in Figure B-10.

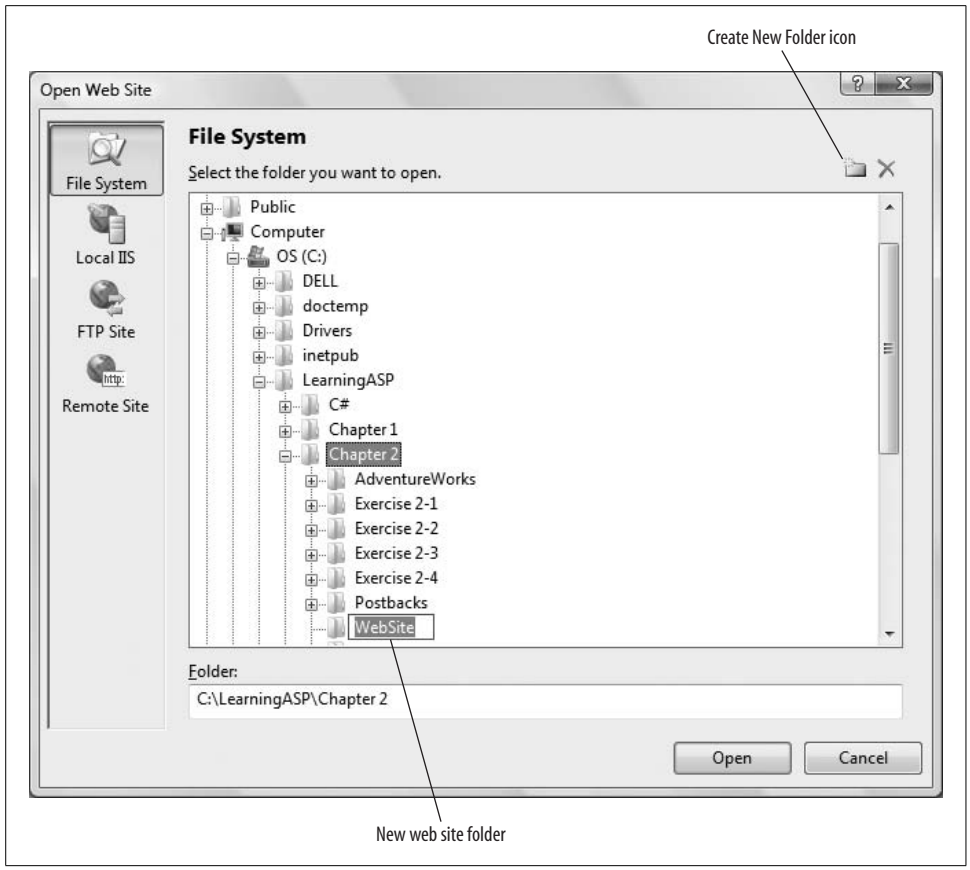


Figure B-8. *In the Open Web Site dialog box, click the New Folder icon to create a new web site folder. You'll be invited to rename it immediately.*

The new web site has now been created and the contents copied over.

To work on this web site, click Open Web Site on the Start page or the File → Open Web Site menu item and select the new web site.

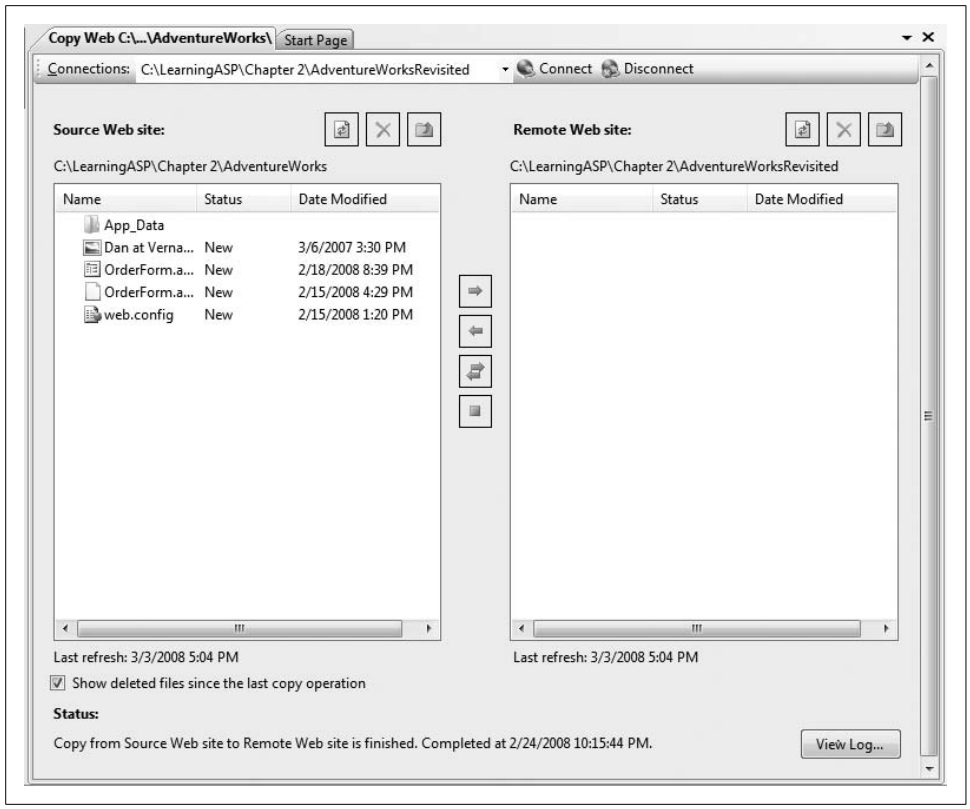


Figure B-9. *After you've created the target web site folder, the Copy Web Site window will show the target.*

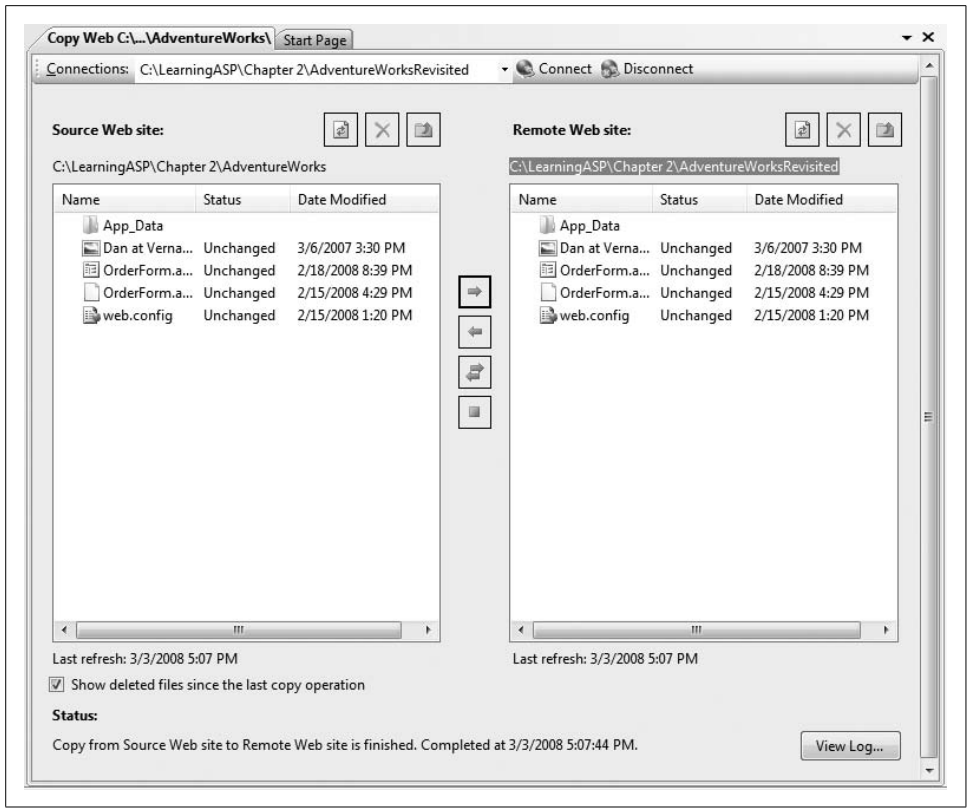


Figure B-10. *After you've copied the web site, both project folders should show the same contents.*

APPENDIX C

Publishing Your Web Site

Throughout this book, you've been developing and testing your web sites on your local computer. You've run all the pages from within Visual Studio 2008, and the web server you've been using is the ASP.NET Development Server that's built into Visual Studio.

This setup is good for testing. However, you can't run a public web site this way. For starters, it's unlikely that your computer is set up to be a public-facing web server. (Or at least, you probably haven't intentionally set it up that way!) In any event, the ASP.NET Development Server that you've been using can't be used to serve a public web site—by design, it accepts requests only from the local computer.

This chapter provides an overview of how you can go about turning the web site you've created on your computer into a web site that anyone can access. You have a couple of options that depend on how much of the site administration you want to take on. The easiest and most common option is to rent space from a company that can host your site.

Another option is to host the web site on a computer you maintain yourself. This is less common, but we'll give you an idea of what's involved in that.

Using a Hosting Company

A *hosting company* or *hosting provider* rents space on their servers where you can upload your web site and keep your data. They will handle the web requests that come to your site, and they'll manage tasks like backup, upgrading the server and .NET Framework software, and so on. Many sites that you've used are on hosting sites—for example, most blogs are on hosting sites. There's no obvious way to tell if a web site is on a hosting site, which is something to remember if the image of your web site (and its URL) is a factor for you.

Every hosting site is slightly different, but they're all similar. In general, they offer a basic hosting package that provides a chunk of disk space and a bandwidth quota, which determines how much data they'll allow your site to send down the wire every

month. Some packages include space on a database server; in other packages, you pay extra for database storage. Many packages also offer email addresses as part of the package, so that you can have one or more email addresses that look like they belong to you—for example, *you@yoursite.com*. (These email addresses are usually what they call *forwarding addresses*—they just forward any incoming email to your normal email account.)

You can expect to pay in the neighborhood of $100 per year for a minimal site, and a little more if you want SQL Server access or if your site gets a lot of traffic. Costs vary somewhat, though, and many hosting sites provide deals for first-time users, so the annual expenditure might vary.

Database Support

An important factor that's different between the test site on your own computer and the hosting site is that most hosters do *not* support SQL Server Express Edition. This is for technical reasons; it's not practical for hosters to run multiple (perhaps hundreds or thousands) of individual instances of SQL Server Express. Almost all hosters therefore instead provide space on a full version of SQL Server. They set up an account for you on their SQL Server and give you the connection string to use to log in to that server, or they provide a way to attach an *.mdf* database file. This difference means that you might have to transfer not just for data that you store as part of your web site, but the data for membership, profile properties, and other information that by default is stored in an *.mdf* file in the *App_Data* folder of your web site.

Your Domain Name

People will access your site by using the site's domain name, like *yourname.net* or *yourcompany.com*. (For more about domain names, see the upcoming sidebar.) For example, one of the authors has a web site that you can access by using the domain name *jesseliberty.com*.

If you get a package from a hosting site, the basic package often lets you make the site available by using a variant on their own name. For example, they might offer you a domain name like *http://www.yourname.theirsite.com*. (If you use the web space that's offered by an ISP such as Comcast, this is what they usually offer.)

If you want a more personal-looking domain name, you can get your own domain name by ordering it from one of the companies that specializes in managing domain names, such as Network Solutions (*www.networksolutions.com*). You buy a domain name for a fixed period—a year, or two, or three. If you have gotten your own domain name, you can register it with the hosting site, and requests to the domain name will go to their servers. If you don't already have a domain name, a lot of the hosting companies will help you with that task as well, and then automatically register it with their site.

Domain Names and IP Addresses

We generally think of URLs as using names like oreilly.com, or asp.net, or something similar. And it's true that when you type in those URLs, you see the pages from those web sites. But the computers that manage traffic on the Internet don't really deal with .com and .net names and all the rest. Instead, they deal primarily with IP (Internet Protocol) addresses. You won't be surprised to hear that, as with most things to do with computers, IP addresses are all numbers.

Try this: type in *http://207.46.19.190*. How about *http://64.233.167.99*? The numbers are IPs, which are the *numeric* addresses for the primary host computers for, respectively, microsoft.com and google.com. The numbers identify a specific location on the Internet. Although this isn't strictly true, you can think of an IP address roughly as the address of a single web server computer out there somewhere on the web.

Obviously, having to remember these numbers is impractical, so the geniuses who thought up the Internet came up with the *Domain Name System* (DNS). Names like microsoft.com are *domain names*. In the DNS, a friendly domain name like *microsoft.com* is mapped to the decidedly less friendly IP number. As part of the superstructure of the Internet, DNS servers maintain lists of domain names and IPs. Simplifying a bit, when you enter a domain name as part of a URL—*http://microsoft.com*, for example—the request goes to the nearest DNS server, which looks up the domain name and gets the IP address, which can then be used to send the request for real. In reality, the volume of requests on the Internet requires a complex system that involves different tiers of lookup, caching, and so on. But in principle, it's a relatively straightforward lookup system.

The benefit of domain names isn't just that it makes it easier for humans to type URLs. It also makes it practical for domain owners to swap computers and even networks—and thus IPs—without breaking URLs.

Anyone can get a domain name. There is a huge central database of all the domain names currently in use. When you want to get your own domain name, you can contact any of the companies that can do the necessary lookup to make sure that the domain name you want is available. For a reasonable fee—from $4.99 to $25.00 per year—you can "own" (actually rent) the domain name of your choice.

Getting the domain name is only one part of the equation, though. The other part is to point that domain name at an IP. This can be a fixed IP address that you control, which is true for big companies with big web presences. For individuals, this isn't as common. If your site is being hosted by a hosting company, you can generally point the domain name to the hosting company's IP address; the hosting company can tell you all about how to do this. If you run your own web server computer, you generally have to run a utility that helps map your computer to your domain name, as explained under the section "Hosting Your Own Site," later in this chapter.

Example: Getting Set Up on a Hosting Site

The clearest way to explain how to get set up on a hosting site is to walk you through the process on a specific hosting site. The details vary by site, but the overall process is the same. To show you how the process works, we'll copy the web site that you created in the exercises for Chapter 10 (the fish aficionados site).

Picking a Hosting Site

You need to select a hosting site that supports ASP.NET. How do you find such hosting sites? One way is to search the web for "ASP.NET hosting." Microsoft also maintains a list of service providers that you can search by country, type, and so on at *http://www.microsoft.com/serviceproviders/directory/default.aspx*. There are certainly a lot of options, but make sure that the hosting site supports ASP.NET 3.5 and ASP.NET AJAX. (Not all of them currently do.) Look through the plans that they offer; one thing you'll find is that some hosting sites are more oriented toward company web sites than toward personal web sites, and offer features and pricing accordingly. Note that any free web-site space that's provided by your ISP probably does not support ASP.NET (or any server-based processing, for that matter). It's always worth checking, though.

For this example, we have decided to use the hosting site DiscountASP.NET (*http://discountasp.net*).

This is not an endorsement of DiscountASP.net over any other hosting company—the authors have also had great success with GoDaddy.com, Orcsweb.com, WebHost4Life, ServerIntellect, and others. But we had to pick one to show you the process.

Setting Up the Account

We began buying a basic hosting package that includes the following:

- Disk space: 1 GB
- Monthly disk transfer (bandwidth): 80 GB/month
- Administrative users: 3

Because we knew we would want to keep data in a SQL Server database, we also ordered the SQL Server 2005 add-on, which provides 300 MB of database storage for an extra monthly fee.

For this exercise, we wanted our own domain name. We therefore got the domain name LearningASPNET35.net, which we were able to order directly through DiscountASP.net, out hosting site provider. This added $15 per year to the cost of the site.

When we finished signing up for the hosting account, we had this information:

- Account name and password. This lets us log in to the hosting site and manage our account, get support, and so on.
- URL for the web site. Because we elected to get our own domain, this is just the URL of the domain name (*http://www.LearningASPNET35.net*).
- An FTP address. FTP (file transfer protocol) is the means by which we'll connect to the hosting site and upload and manage our files.
- The name of the SQL Server on the hosting site, the name of a database on that server, database login credentials, and a connection string that we can use in our web site to access the database.

Uploading the Web Site Files

Uploading files from the local computer to the hosted site is easy—you can use the same Copy Web Site utility that you've been using throughout the book. One difference, though, is that you will connect to the hosting site by using an FTP connection. If you are not familiar with FTP, it's a way to connect to a remote server and copy files to and from that server. In practice, you can use FTP a lot like you use Windows Explorer, except that you have to log in first.

In the Copy Web Site window, click Connect. In the Open Web Site window, click the FTP Site tab on the left. The window changes to show the fields that you fill in to connect to an FTP server. In the server box, enter the name that the hoster has provided for the FTP server. Uncheck the Anonymous Login option and then enter the user name and password for your hosting account. Leave the Passive Mode option selected.

Figure C-1 shows what the Open Web Site dialog box will look like.

When you've connected successfully, you will see the familiar Copy Web Site layout, with your local files on the left, and the remote server (which really is remote this time) on the right. Select everything from your local web site *except* the App_Data folder, and then click the arrow key to copy everything to the FTP server. (We'll get to the App_Data contents in a moment.) Figure C-2 shows the Copy Web Site dialog box ready to copy files up to the FTP server.

That's it—you've copied your ASP.NET web site files to the server. Except for the data in the database (membership and profiles), you're ready to go.

Uploading Data

A somewhat more challenging task is to move data from your local computer, where you've been using SQL Server Express, to the SQL Server database provided by the hosting company. As noted earlier, hosters generally do not support SQL Server

Figure C-1. *The Open Web Site dialog box, showing how you connect to an FTP server. (Sensitive information in the picture has been obscured.)*

Express. They *might* let you attach an *.mdf* file that is in your *App_Data* folder, but that's up to the hosters. For questions about database access on a hosting site, you'll need to dig into their documentation and support forums a bit.

For this example, we're assuming that we need to use the database that they have set up for us. In the example web site, we set up user accounts (membership) and profile properties, which automatically created the *ASPNETDB.MDF* database in the *App_Data* folder. We want the test information that we created in the *ASPNETDB.MDF* database to be available on the hosted site as well. However, we cannot directly copy the database tables and data to the hosting site's SQL Server.

A solution is to use the Database Publishing Wizard. This wizard reads through your existing SQL Server Express database and creates a *SQL script* (a *.sql* file) that contains a list of the SQL commands that are required to re-create the database you already have. This script can be quite hefty—the script that's generated in order to recreate the tiny database we have for the example is over 5,000 lines. (Good thing that this is all done automatically!) When the *.sql* file has been created, you log in to

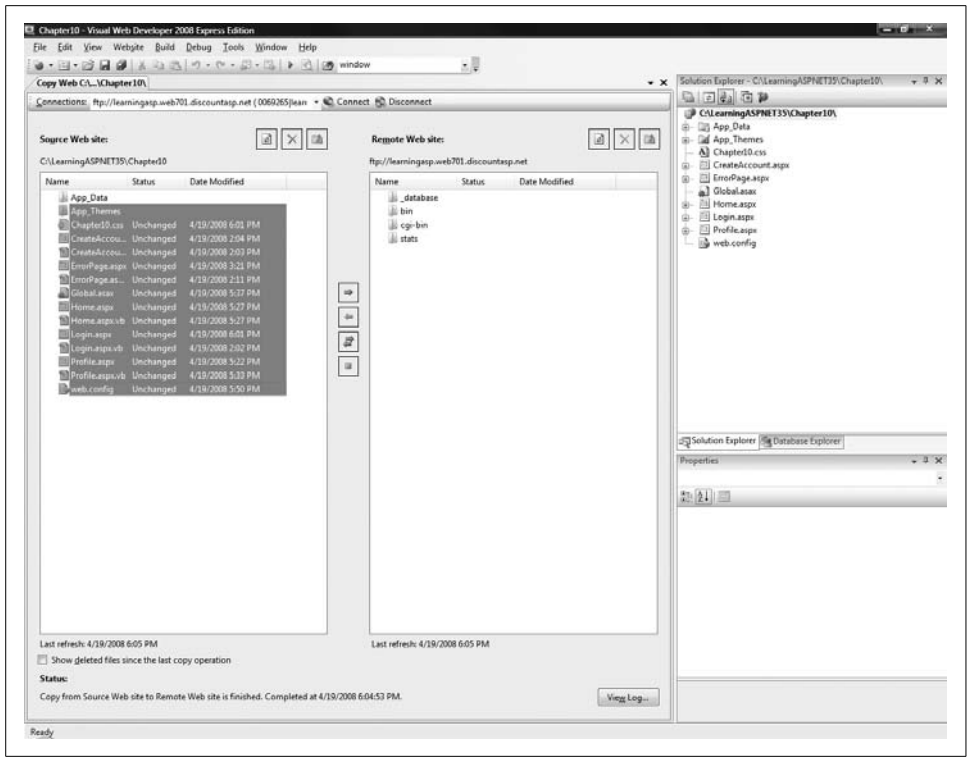


Figure C-2. *The Copy Web Site window, ready to copy the contents of the web site (except the App_Data folder) to the hosting server.*

the hoster's database and then run the generated script. When the script has finished, a copy of the database is available on the hosting site.

There are a few additional tasks to perform as well. We'll walk through the entire process, which includes the following:

- Creating the script.
- Logging into the hosting database and running the script.
- Updating the connection strings in your web site to point to the database on the hosting site.

By the way, we strongly encourage you *not* to perform these steps with the AdventureWorks database that you've been using throughout the book. That's strictly a database for testing, so it's not useful on a live site. Moreover, the database is huge, which would up eating a significant part of your allocated database space.

Creating the script

Creating the script is straightforward. In Visual Studio 2008, display the Server Explorer window (Database Explorer in Visual Web Developer Express Edition). In our case, under Database Connections, we have a connection to the ASPNETDB.MDF database.

Right-click the database that you want to create the script for (that is, to "publish") and then click "Publish to provider," as shown in Figure C-3.

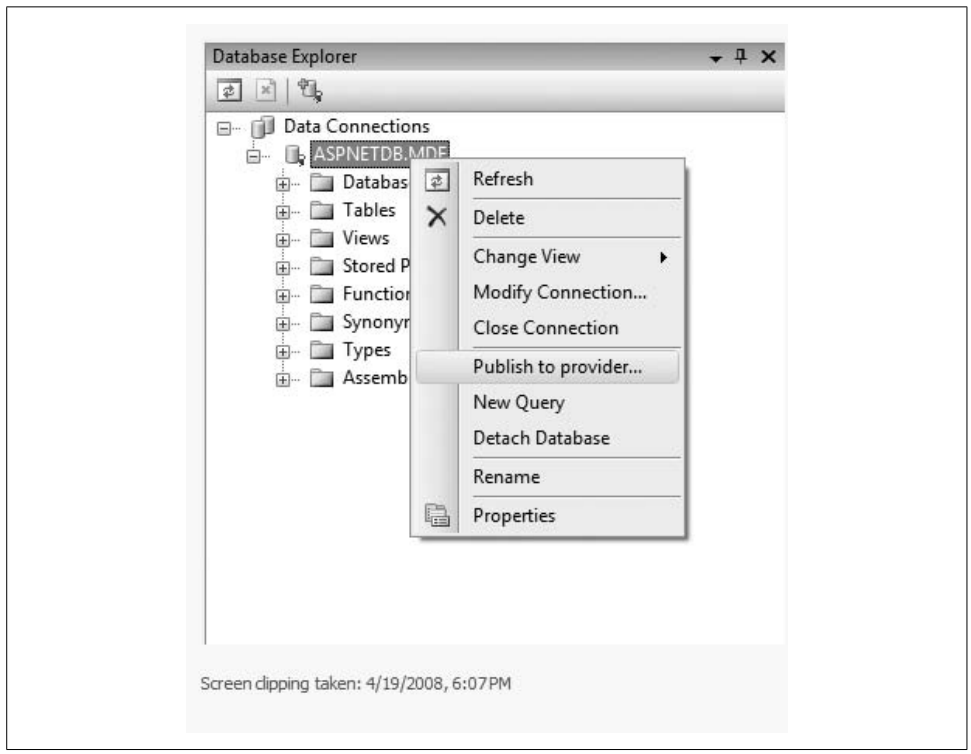


Figure C-3. *Beginning the process of creating a SQL script file for a database.*

The Database Publishing Wizard starts up.

Click Next, and then select the database you want to publish. For us, this means selecting the *.mdf* file that contains the membership and profile information for the web site we have uploaded. Figure C-4 shows this selection.

Click Next, and then specify a target location for the *.sql* file, as shown in Figure C-5. You will notice that there is an option in this page of the wizard to specify a "shared hosting provider." Some hosting providers make their database available over the web by using a web service. If your provider does this, you can copy data directly from your local database to their database. Check with your hosting provider about

Figure C-4. *Selecting the database to publish.*

this capability. Our provider appears not to offer this feature, so we will proceed with creating the *.sql* file.

Click Next. The next page lets you specify options for creating the script. The term "drop" is used in SQL Server to mean "delete"; here, they're asking you if you want to delete existing tables in the target database before creating new ones. We left the default of true. (You will want to be very careful with this if you are uploading to an existing database—if you select the option to drop existing objects, they will be permanently deleted.) "Schema qualify" asks whether you want to add the database schema names to the new database—remember the adventures you had with database schemas in Chapter 4. We left this as true. For target database, make sure that the target matches the database version that your hoster provides. And finally, select whether you want to copy only the database structure (schema) or the structure and the data that it contains. If your local database contains only throwaway data that you don't need in the production web site, you can copy just the schema. If your local database already contains live data, copy the data as well. Figure C-6 shows this page of the wizard. (In fact, for this exercise, we could copy just the schema, because it would be easy for us to re-create the few users and profile properties that we

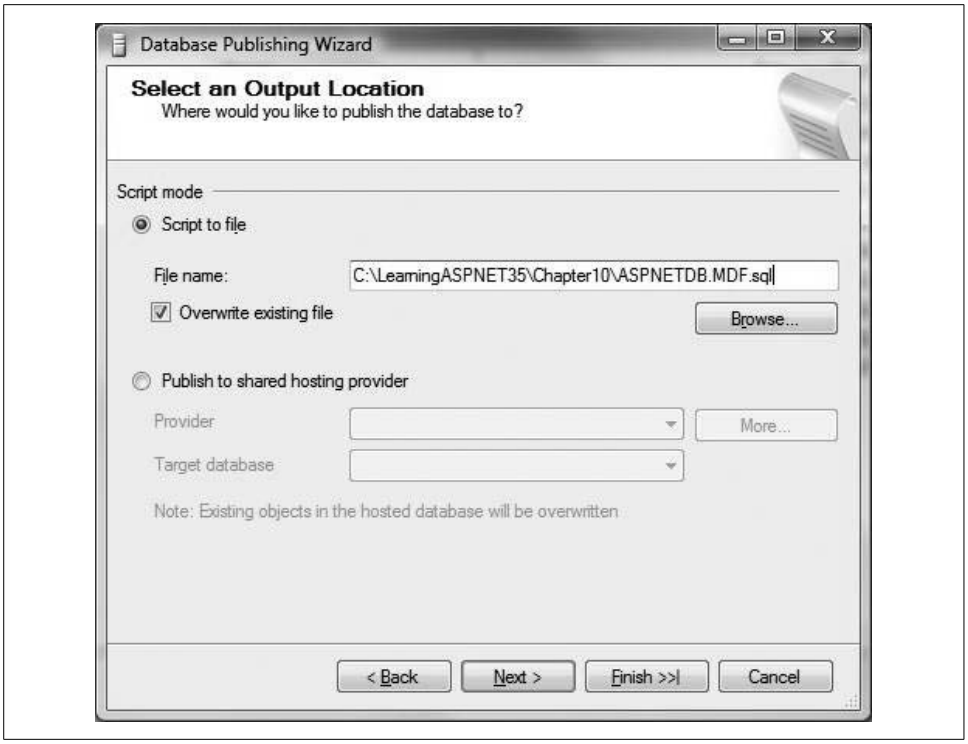


Figure C-5. *Selecting a target location for the .sql file.*

created in order to test the web site. However, we will go through this exercise as if we had data that we absolutely needed to copy to the hosting site.)

Click Finish. The wizard gets to work creating the SQL script file, as shown in Figure C-7. The process can take a minute or two, depending on the number of tables and the amount of data in each table.

When the wizard has finished, click Close.

The end result is a *.sql* file in the location that you specified. If you're curious about what the script looks like, open the file in a text editor like Notepad. (But don't change anything!) Now you're ready to re-create the database on the hosting site.

Logging Into the hosting database and running the script

Your hosting provider will provide some way for you to manage your database tables. A common way is to let you attach to their database server by using a SQL Server management console. If you have a full version of SQL Server installed on your computer, you already have this tool. If you are working with SQL Server Express Edition, you probably do not. In that case, you can download and install the

Figure C-6. *Setting options for what the .sql script will do.*

free SQL Server Management Studio Express tool from the Microsoft web site at *http://msdn2.microsoft.com/en-us/express/bb410792.aspx*.

We'll proceed with the idea that the hoster lets you use SQL Server Management Studio Express. (Again, you must check with them to get the details of what they provide.) Start the management console—from the Windows Start menu, click All Programs → Microsoft SQL Server 2005 → SQL Server Management Studio Express.

You're prompted to connect and are asked for the database server name and the login credentials. You will have gotten these from the hosting company during the signup process. Figure C-8 shows the login screen for the management console.

Click Options, and then in the Network protocol list, select TCP/IP, as shown in Figure C-9.

Click Connect. If you've got the correct credentials and if the network protocol works, you'll be connected to the hoster's SQL Server.

Figure C-7. *The wizard as it generates the .sql file.*

Figure C-8. *Connecting to a SQL Server using SQL Server Management Studio Express.*

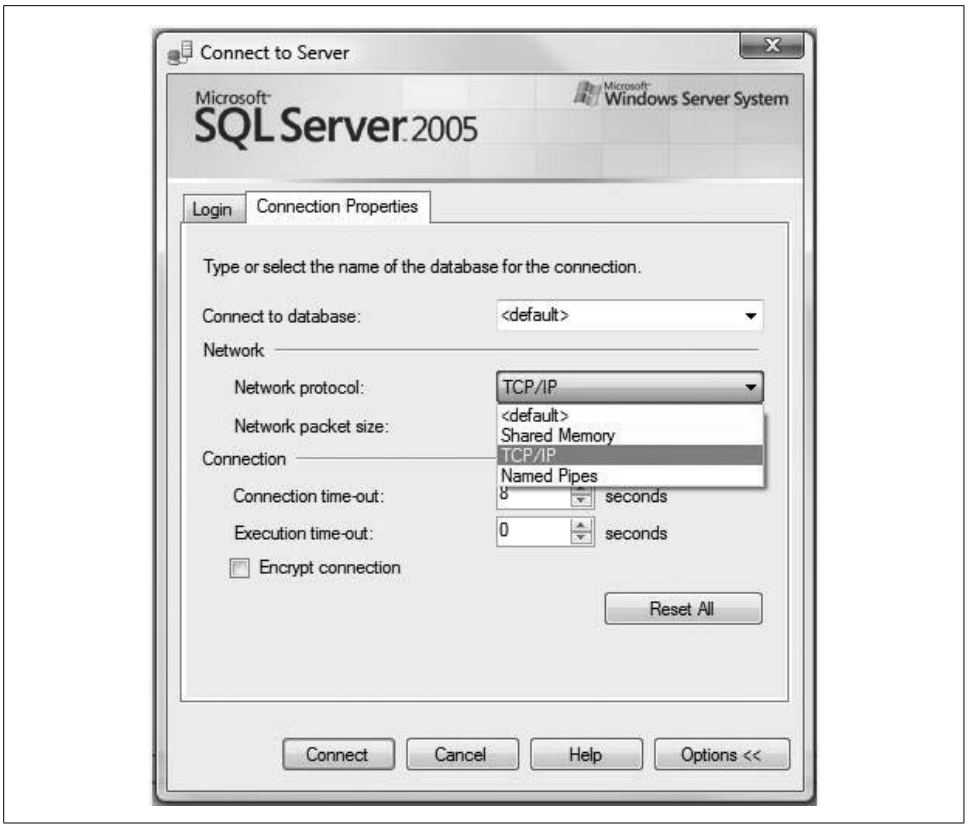


Figure C-9. *Setting the network protocol for connecting to a SQL Server database.*

If you cannot connect to the database, check all the information again and make sure that you've got the database name and credentials right—they might not be the same as the information for logging onto the hoster's web site. If you still have no luck, contact your hosting provider and get their recommendation for how to establish the connection.

After you've connected, on the left-hand side, under Object Explorer, find the database that's been created for you and then expand it. Figure C-10 shows what this looks like for our hosting provider's SQL Server. On our hoster's SQL Server, ours is just one of hundreds of databases, so we had to scroll quite a bit to find our particular database.

Now you can run the SQL script that you created earlier. In the top-left corner, click New Query. This opens a new tab on the right where you can type (or copy) SQL statements. Using a text editor, open the *.sql* file that was generated, copy the entire contents, and then paste the text into the query tab. When you're ready to go, click Execute in the toolbar.

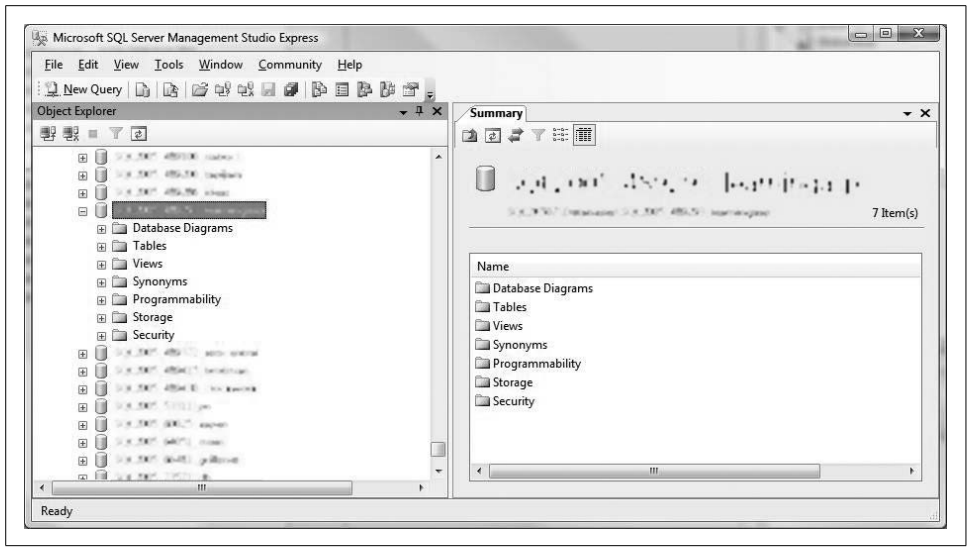


Figure C-10. *After you have connected to the hoster's SQL Server, you can open your database.*

It will take a minute or two for the SQL Server to crank through the script. When the script is finished, the results are displays in a Results window. If all went well, you will see line after line that says (`1 row(s) affected`), as shown in Figure C-11.

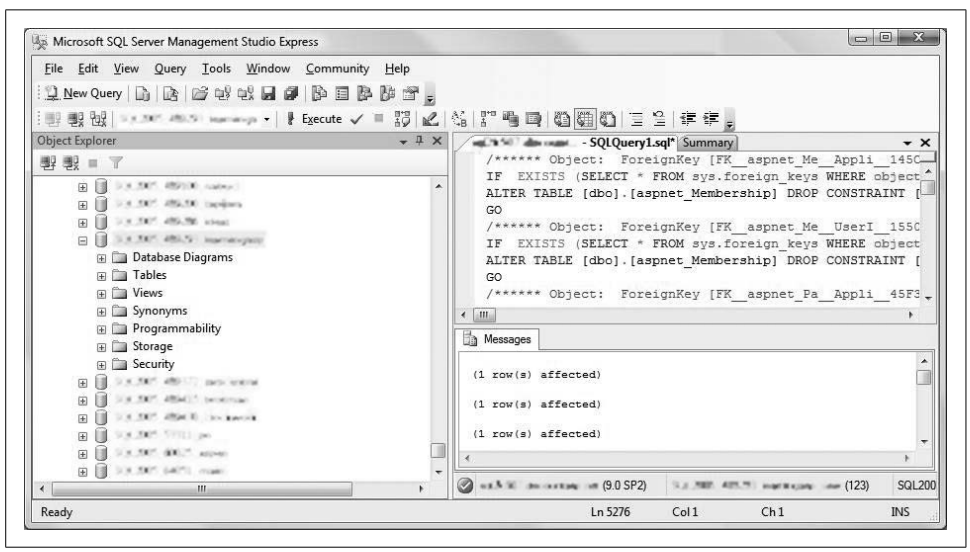


Figure C-11. *When the script finishes, the results are shown in the Messages tab. A series of lines that say "(1 (row(s) affected)" indicates success.*

If the script does not run successfully, you will need to study the error messages. If you can't determine the problem, you will need to contact the hosting provider for support. (Our apologies if this suggestion sounds like a stuck record. Unfortunately, because each situation is different, it's difficult to help much with troubleshooting the many possibilities for each problem.)

If you did have success, your database and data are now on the hosting provider's SQL Server. Only one step is left, which is to update the connection string or strings in your application so that they point to the correct database.

Updating the Connection Strings

Now that the data is in a new location, you need to let your web site know where to find it. This is where it comes in handy to keep database connection string information in the *web.config* file, as we illustrated in Chapter 4.

In Visual Studio, open the web site (if it isn't already open) and make a copy of the web.config file. Name the copy something like *web.config.local*—this will be the version of the *web.config* file that contains settings for testing the web site on your computer. Then, open up the original *web.config* file.

Look for the `<connectionStrings>` element. If the web site you are uploading involved database access other than the ASP.NET membership and profiles database, you probably already have one or more connection strings already defined, like this:

```
<connectionStrings>
  <add name="SampleDatabaseConnection"
    connectionString="[connection string information]"  />
</connectionStrings>
```

In that case, find the connection string information that your hosting provider sent you, and substitute that information for the value of the `connectionString` attribute. The hoster's connection string will probably have placeholders where you need to include an explicit user name and password.

If you did not explicitly create a connection string for this web site, the element will be empty, like this:

```
<connectionStrings />
```

In that case, re-create the element with separate opening and closing tags, like this:

```
<connectionStrings>
</connectionStrings>
```

For both cases (whether there was an existing connection string or not), you'll need to add a connection string that tells ASP.NET where to find the membership and profiles database. Create a new child element that looks like the highlighted part of the following example:

```
<connectionStrings>
  <!-- Any existing connection strings here, updated with the new
       connection string from the hosting provider -->
  <add name="LocalSqlServer"
    connectionString="[Connection string from hosting provider]"/>
</connectionStrings>
```

Notice that the `name` attribute of the new element is `LocalSqlServer`. This is the connection string name that ASP.NET uses to find the membership and profiles database. By default, you don't have to define this because it's predefined to point to the *ASPNETDB.MDF* database in the *App_Data* folder of your web site. However, you have now moved that database, so you need to create a connection string that has exactly this name (`LocalSqlServer`) and that provides the alternative connection string information. As is suggested in the example, the `connectionString` attribute should be set to the connection string that the hosting provider has given you. If necessary, substitute your own user name and password for the placeholders in the hosting provider's connection string.

Save the *web.config* file. Then use the Copy Web Site utility to copy the updated version of the *web.config* file to the hosting site.

Run the web site using the URL for the site and the home page. (For us, that's *http://www.learningaspnet35/home.aspx*.) Did it work? If so, congratulations! You have a live web site on the Internet.

Notes and Tips

Here are a few tips for you about working with your web site that's hosted elsewhere.

In Visual Studio and VWD, you can open a web site directly over an FTP connection. To do so, choose File → Open Web Site. Click the FTP tab on the left and provide FTP login information, as shown previously in Figure C-1. When the web site opens and is displayed in Solution Explorer, you are working with the live web site. However, there are a couple of reasons why you might not want to do this. One is that working with a web site directly over FTP can be slow; Visual Studio not only has to copy files back and forth between the server and your computer, but it maintains a local cache of files. Another reason you might not want to work directly over an FTP connection is that you are then working with the live site, and if you break something, it's broken for all your users as well. Doing maintenance and testing directly on a live site is not a good idea!

As suggested in the procedure earlier, keep two copies of the *web.config* file, one for testing locally and one for deployment to the live site. It can be tedious to keep these in sync (except for where they must differ, of course), but it's better than changing a single *web.config* file back and forth.

Turn off custom errors until you've got everything working. Otherwise, when you're testing your newly live site, you won't be able to see details about errors. When you've got everything working, though, turn custom errors back on. For details, see Chapter 8.

Be sure to turn off any debugging settings on the live site. In the *web.config* file, make sure that the `<compilation>` element is set like this:

```
<compilation debug="false" ... >
```

Make sure also that in the `@Page` directive on individual pages, you have set `debug="false"` or removed the debug attribute altogether.

If you are not using the *Default.aspx* page for the site (for example, your home page is *Home.aspx*), create a simple *Default.aspx* page that redirects to the home page, using code like the following:

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    Response.Redirect("Home.aspx")
End Sub
```

IIS, which is the web server software that your hosting company uses, is configured to look for a *Default.aspx* page on any ASP.NET site. You can take advantage of that to redirect from the *Default.aspx* page to the page that you want users to see first, even if they enter just your domain name without any *.aspx* file in the URL.

Remember that any time you make a change to any page or file in your site, you'll need to connect to the hosting site and upload the new version. This sounds obvious, but it's not that hard to forget, and you can spend a surprising amount of time trying to debug what seems like a problem but turns out to be that the live site has an old copy of a file.

Finally, be sure to test changes immediately and make sure that you test them thoroughly. Don't let your users find problems for you.

Hosting Your Own Site

You can host a site on your own computer. Hosting your own site is not a job for a complete beginner, but it's not rocket science. You need the following:

- A computer and an operating system that can run web server software, specifically Microsoft Internet Information Services (IIS).
- An "always-on" connection, like what you get with cable-modem, DSL, or fiber service. (It might not need to be said, but you can't really run a web site on a computer that uses dial-up to connect to the Internet.)
- A registered domain name.
- A fixed IP address for the server computer, or alternatively, a dynamic DNS redirection service. (This probably sounds like gibberish, but we'll explain it shortly.)

Running Web Server Software

A web server is not a computer as such; it's a program that runs on a computer, listening for web requests and fulfilling them. If you want to host your own web site, you need to run web server software on a computer that you can make available to the Internet.

For your purposes, web server software means Microsoft IIS. This is the server software that ASP.NET is designed to run on. When you host a site, IIS is playing the role that the ASP.NET Development Server has played when you've tested your web sites.

IIS is available on Windows XP, Windows Vista, and of course, on the Windows Server products—Windows Server 2003 and Windows Server 2008. For security and performance reasons, it is not installed by default on any of these products; you must add it as a Windows feature in the Control Panel. There are two versions of IIS, depending on which operating system you have. If you have Vista or Windows Server 2008, you run IIS 7. If you have Windows XP or Windows Server 2003, you run IIS 6.

If you run IIS 7 (that is, you have Vista or Windows Server 2008), you need to explicitly configure support for ASP.NET. For details, see the topic "Running Web Applications on Windows Vista with Visual Studio 2005" on the MSDN web site at *http://msdn2.microsoft.com/en-us/library/aa964620(vs.80).aspx*.

If you have only one computer, you can run IIS and SQL Server on it and use that computer as your web server. However, this will put a load on your computer. Web server processing will affect the performance of your other programs. Conversely, if you run processor-intensive and disk-intensive programs, they will affect the performance of the web server and the database. This might be OK if your web site has very light traffic, but for web sites of any size, you should probably have a dedicated computer to run IIS on.

Let's assume for the remainder of this section that you decide to set up a spare computer with Vista, and you install IIS, SQL Server Express Edition, and the .NET Framework version 3.5 on it. Now what?

Security

The first thing to do is to *do your best to make your web server computer secure*. If you run your own web server, you are inviting the entire Internet into your home and onto your server. It is *critical* that you take precautions to protect the server and any computers that it is attached to. These steps include, but are not limited to, the following:

- Make sure that the web server computer is formatted using the NTFS file system. Using the alternative FAT system does not provide enough security for individual folders and files.
- Put the computer behind a firewall. A typical approach is to use a router that has a built-in firewall and to put the server computer behind it. Review the firewall settings to ensure that ports are all closed except port 80, which is used for web traffic. (If you have other computers behind the firewall, you might need to have other ports open for functions such as IM—however, keep the open ports to the few that you absolutely have to have.) An easy way to check the status of your ports is to use Steve Gibson's Shields Up! utility. (Search for it on the Web.)
- Install the absolute minimum amount of software on the server computer. Ideally, you will have only IIS and SQL Server. (You can use SQL Server Express if you are hosting a site on your own computer.) The less software there is, the fewer security vulnerabilities you have (and the better the performance of the server computer).
- Remove any user accounts that you might have created that have administrative privileges. Note that ASP.NET and IIS create user accounts like `ASPNET`, `IUSR_<name>`, and `IWAM_<name>`. These are required, so leave them; the good news is that they have limited permissions.
- Be fanatic about installing the releases that are part of Windows Update. These releases frequently include security patches that help keep down the vulnerability of your web server and of SQL Server.
- Be fanatic about backing everything up, using multiple generations of backup. If your server is compromised (or just plain crashes), you can restore from the most recent good backup.

Sound scary? It is. The sad fact is that your computer or computers are being probed continually for security vulnerabilities by malicious users out there. They don't have it in for you personally. They're just constantly on the lookout for computers with weak defenses that they can compromise, so that they can take control of your computer and use it for their own purposes—as a spam forwarder, as a "zombie," and so on. Don't think that because your site is small and not very important, your server is not worth anyone's trouble. It is.

Configuring the Firewall

If your server computer is sitting behind a firewall (as is strongly recommended), you will need to make a few changes to how the firewall is configured. For starters, you must make sure that port 80 is open in the firewall. This is the default port for web traffic, and in order to receive web requests, your server must have that port open. In Windows Firewall, this means setting up an exception for port 80, since by default virtually all ports are closed. You might find that in Windows Firewall, port 80 isn't

always referred to by its port number. For example, it might just be called "World Wide Web Services (HTTP)" as in Figure C-12.

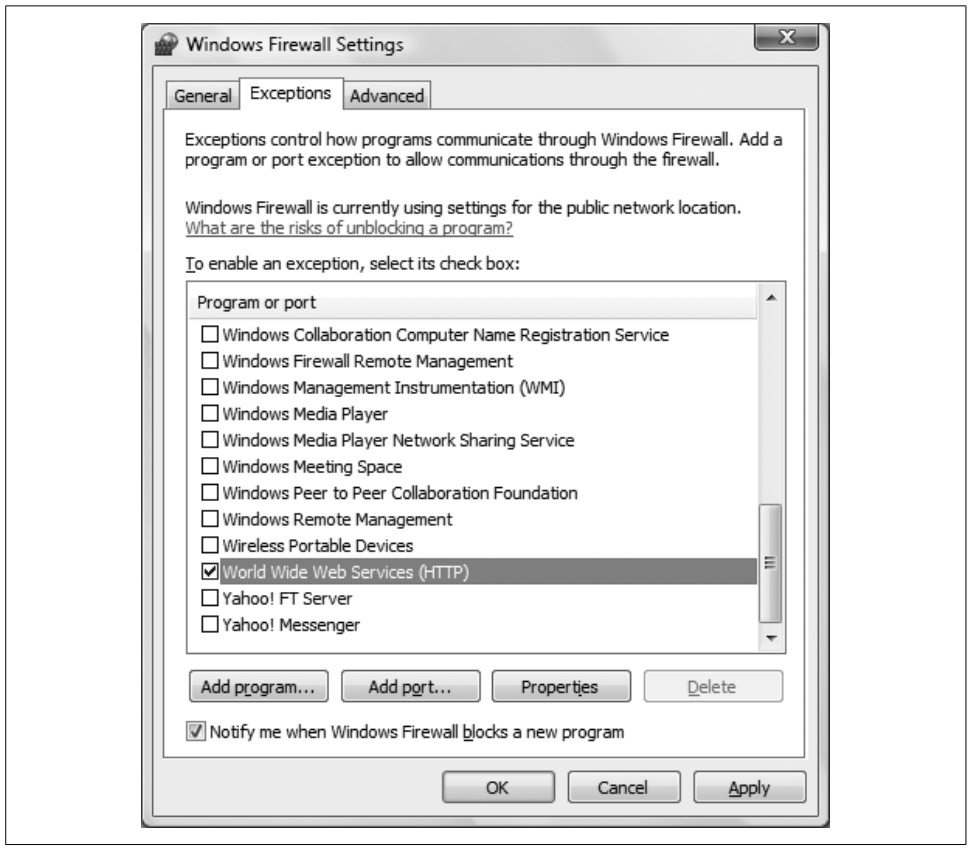

Windows Firewall Settings
General
Exceptions
Advanced
Exceptions control how programs communicate through Windows Firewall. Add a program or port exception to allow communications through the firewall.
Windows Firewall is currently using settings for the public network location.
What are the risks of unblocking a program?
To enable an exception, select its check box:
Program or port
Windows Collaboration Computer Name Registration Service
Windows Firewall Remote Management
Windows Management Instrumentation (WMI)
Windows Media Player
Windows Media Player Network Sharing Service
Windows Meeting Space
Windows Peer to Peer Collaboration Foundation
Windows Remote Management
Wireless Portable Devices
World Wide Web Services (HTTP)
Yahoo! FT Server
Yahoo! Messenger
Add program...
Add port...
Properties
Delete
Notify me when Windows Firewall blocks a new program
OK
Cancel
Apply

Figure C-12. *Opening port 80 in Windows Firewall to allow incoming web server traffic.*

If the computer is behind a router that includes a firewall, you will also need to configure *that* firewall to let port 80 traffic through. It is likely that you will have to additionally set up *port forwarding*. In port forwarding, the router accepts traffic on a specific port and then forwards the traffic to the same port on a single computer on the network behind the router. (Which is, of course, your server computer.) The instructions for configuring ports and port forwarding for a hardware firewall differ substantially between routers. About the only thing they have in common is that they usually provide a way to access their configuration pages from a browser, usually on an address like *http://192.168.1.1* or something similar. Beyond that, however, the best we can tell you is to consult the documentation that came with the router. If you don't have the documentation, you can probably find it online at the router manufacturer's web site.

Hosting a Web Site Under IIS

After you've set up the web server, you can publish your web site to IIS. The easiest way is to copy your web site files to a new folder on the web server computer, and then to create an *IIS application* that points to the folder.

In IIS terminology, an "application" is roughly equivalent to what we've been calling a "web site" in Visual Studio and VWD, leaving aside some technical details. In IIS, a "web site" is the entire site—in simple terms, *yourdomain.com*. This is an unfortunate naming conflict that just means that you have to mentally switch gears when moving between IIS and VWD.

If your development computer and the web server computer are on the same network, you can usually copy the files across the network (using the Copy Web Site window, if you like). Otherwise, you'll need to transfer them on a CD or via some other transportable means.

Make sure that the web server computer is running the .NET Framework version 3.5.

The steps you take to create an IIS application depend on what version of IIS you are running. The steps in IIS 7 and IIS 6 are similar, but the UI is different. We'll show you screenshots only for IIS 7, but we'll provide instructions for both.

IIS 7

On the web server computer, you use Internet Information Services (IIS) Manager to create and manage web sites. To open the management console, from the Windows Start menu, choose Run and then type `inetmgr`. The manager window is displayed, as shown in Figure C-13. (If you are running Vista, Vista might ask your permission to continue.)

Open the node for the local computer, and then open the Web Sites node. Right-click Default Web Site, and then choose Add Application, as shown in Figure C-14.

The Add Application dialog box is displayed, as shown in Figure C-15. For alias, fill in the name that you want your application to have in the URL (for example, *http://yourdomain.com/alias*). Fill in the path to the folder where you've put your web site files and then click OK.

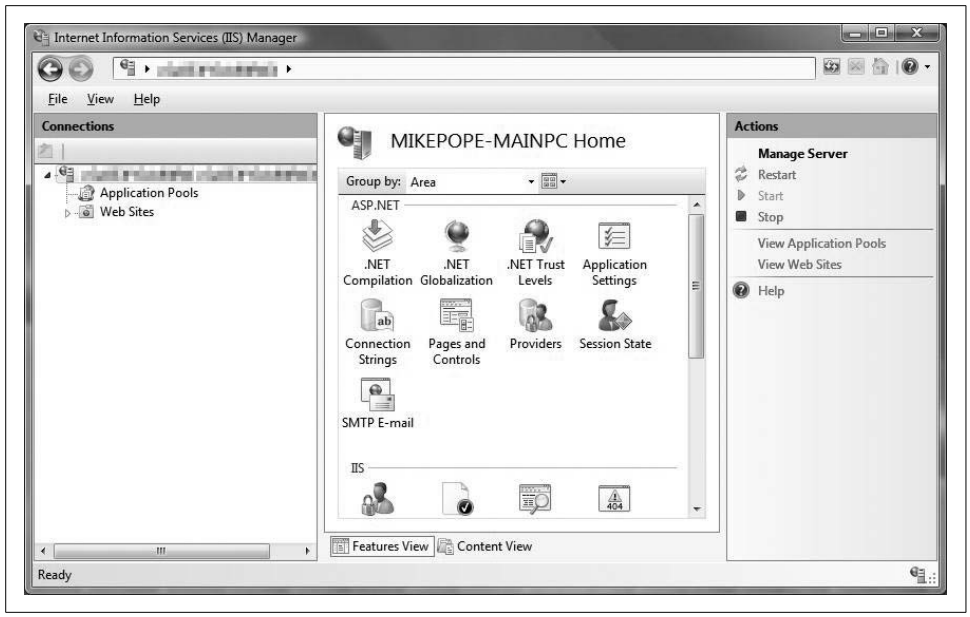


Figure C-13. *The IIS management console, where you create and manage web sites on the web server computer.*

To test your site, open a browser and type *http://localhost/alias/*. If everything is working, the *Default.aspx* file for your application will run. If it isn't, here are some things to check:

- The files are in the folder
- The IIS application points to the correct folder
- ASP.NET is enabled for IIS 7

If you're stumped, do a web search for the exact error message you get and see how the problem was resolved. (It's a safe bet that you won't be the only one who's seen the error.)

IIS 6

In IIS 6, the story is similar. To start the management console, type `inetmgr` in the Windows Run box. Open the node for the server computer, and then open Web Sites. Right-click Default Web Sites and then choose New → Virtual Directory. This starts the Virtual Directory Creation Wizard.

In the wizard, enter an alias and the path of the folder that contains your files. In the Virtual Directory Access Permissions pane, select the Read, Run scripts, and Execute permissions. (Setting these permissions causes IIS 6 to automatically create an IIS application for your virtual directory.) Finish the wizard and then test your application by opening a browser on the server and typing *http://localhost/alias/*.

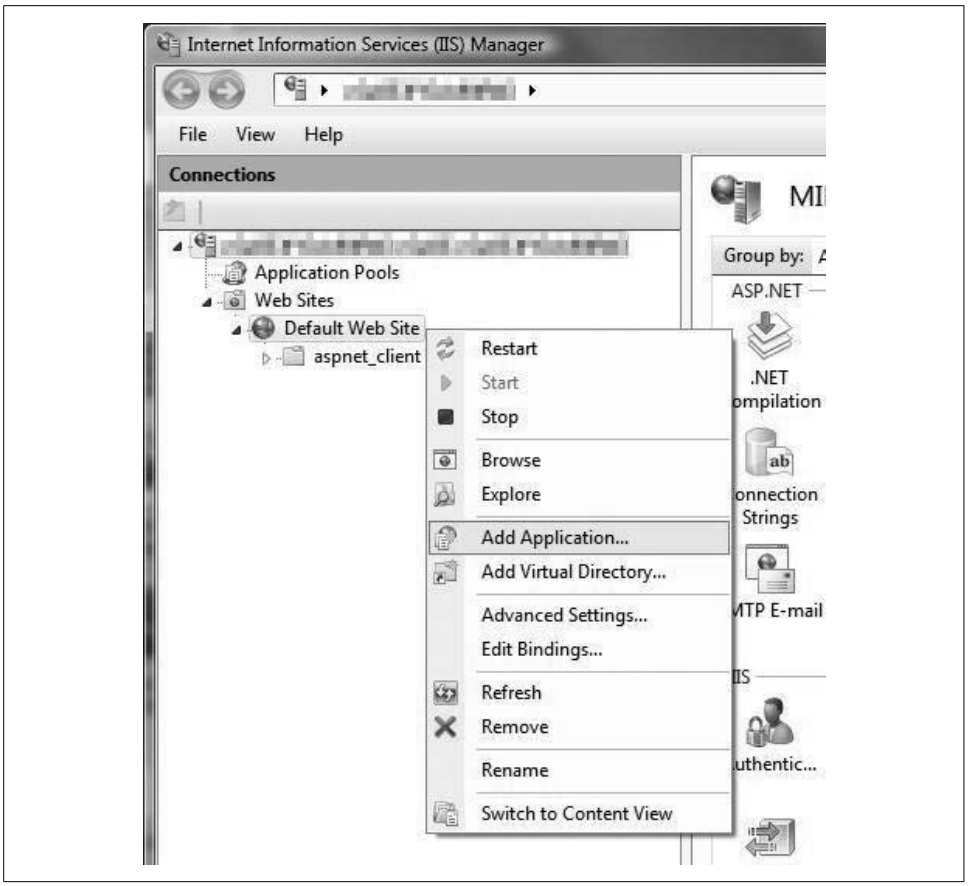


Figure C-14. *Creating a new IIS application for the default Web site in IIS 7.*

Setting Up the Domain Name

A final step is to get a domain name and then to point the domain name to the IP address at which your server computer can be reached. For background on domain names, see the sidebar, "Domain Names and IP Addresses," earlier in this chapter.

If you want to host a web site on a computer that you have at home, there's a bit of a wrinkle in getting the domain name to point to your IP address. Unless you have paid for a fixed IP address, your computer probably does not have one. Instead, you probably get your IP address through your ISP, which has a big pool of IP addresses and which assigns one of those to your computer whenever you log on to their network. In that case, you have a *dynamic IP address*. A dynamic IP address makes things slightly tricky for purposes of registering a domain name to that address.

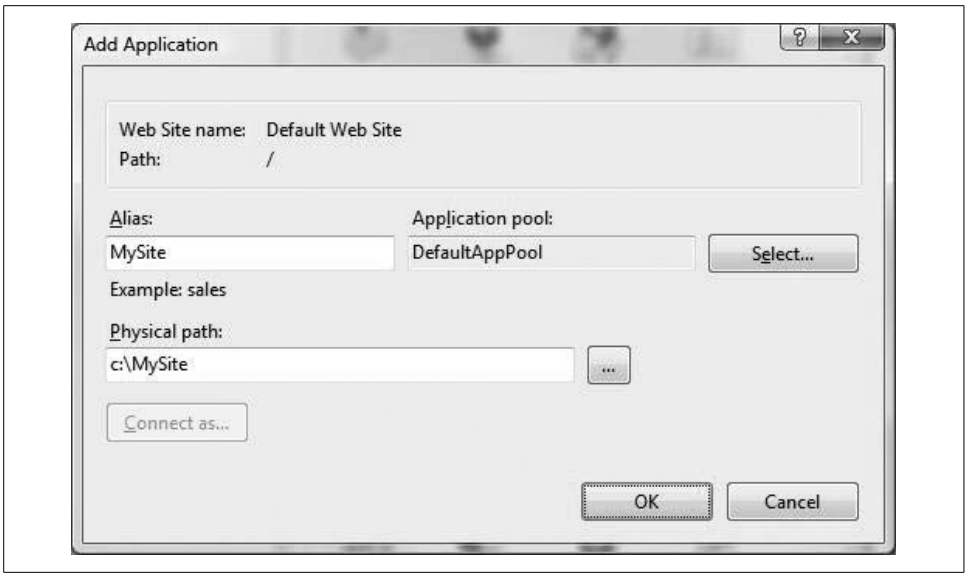


Figure C-15. *Creating an IIS application for the folder where your web site files are.*

The solution is to use something referred to as a *dynamic DNS redirection service*. It works a little bit like a hosting service—the redirection service company has servers with fixed IP addresses. You register your domain name with them, using their IP address. Then, on your server computer, you run a utility as a Windows service that periodically checks what IP address your ISP has currently assigned to you. This utility sends your current IP address to the redirection service at intervals (as often as every minute), so that the redirection service knows where the domain *really* is.

When a request is made to your domain name, it initially goes to the IP address for the dynamic redirection service. The server at the dynamic DNS redirection company in turn looks up your domain name in its database, finds whatever your IP address was as of the last time your computer phoned home, and sends you the request. The web server on your computer then handles the request and sends the response.

If this is your situation, you can investigate domain redirection services by searching for "dynamic DNS redirection" on the web.

Further Resources

In this appendix, we've given you just an overview of how you can publish your web site to the world. As noted, hosting sites differ in what they offer and in how you work with them. If you opt to use a hosting site, take advantage of the extensive help and user forums that all the hosters offer to get more information about managing your account.

The lists below provide more information about how to publish a web site. If you are using a hosting company, see the following:

- The support forums for the hosting company that you select.
- Scott Guthrie's blog, which includes posts about deployment, about the Database Publishing Wizard, and more. Search his blog (*http://weblogs.asp.net/scottgu/*) for topics like "Database Publishing Wizard."

If you are running your own web server computer, see these resources:

- The *http://iis.net* site, which provides information and support for IIS.
- "Securing Your Web Server," an in-depth article from the Microsoft Patterns and Practices group, available at *http://msdn2.microsoft.com/en-us/library/aa302432.aspx*.

APPENDIX D

Answers to Quizzes and Exercises

Chapter 1: Getting Started

Answers to Quiz Questions

1. You can create a new web site by selecting File → New Web Site, or by clicking the "Create: Web Site" link on the Start page.
2. The three views are Design view, which shows you the appearance of your page, Source view, which shows you the markup, and Split view, which shows both.
3. The settings that are specific to each control are called properties, and you can view them in the Properties window of the IDE.
4. The controls are kept in the Toolbox, which is on the left side of the IDE by default.
5. There are three different ways to run your program with debugging: click Debug → Start Debugging, press F5, or click the Start Debugging icon on the toolbar.
6. The `Click` event.
7. The code for the event handler is located in the code-behind file.
8. You can get to the code-behind file by selecting the file in the Solution Explorer, or by double-clicking the control whose default event you want to set up.
9. Use the `Label` control's `Text` property to set the content of the label. You'll see that many controls have a `Text` property that you can set.
10. When you click the button in Hello World, you raise the `Click` event. The code for the event handler is evaluated at the server, which redraws the page and sends it back to the browser, causing a noticeable flicker. In Chapter 2, you'll see how to avoid that.

Answer to Exercise

Exercise 1-1. There is no "right" or "wrong" answer to this exercise; you're just playing around with the properties to get a feel for the range of options you have. If the changes you're making aren't showing up properly, make sure you've selected the label control; you can tell because "Label1" will be displayed at the top of the properties window. If you're having difficulty changing the text of the label, be sure you're looking in the `Default.aspx` markup file, which you can access by clicking the tab at the top of the page, or through the Solution Explorer.

If you switch to Source view, you'll see the properties you've assigned reflected in the markup, and you can change the values here as well, of course. Example D-1 shows the markup for one author's page.

Example D-1. *One version of the markup for Exercise 1-1*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb" Inherits="_
Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:Button ID="Button1" runat="server" Text="Button" />
        <asp:Label ID="Label1" runat="server" Text="Label"
                   BorderColor="#0033CC" BorderStyle="Dashed"
                   Font-Bold="True" Font-Size="Larger"
                   ForeColor="#0033CC"></asp:Label>
    </div>
    </form>
</body>
</html>
```

Chapter 2: Building Web Applications

Answers to Quiz Questions

1. A postback is when the page is returned to the server to evaluate code handlers, and the same page is sent back to the browser afterward.
2. The first type of postback in ASP.NET is synchronous, in which the entire page is sent back to the server, as with a non-AJAX page. The second type, with AJAX, is asynchronous, in which only part of the page is updated, and the rest is unaffected.

3. The ID property. The IDE sets this property for you (although you can change it), and you need it to refer to the control from elsewhere on the page.
4. Use a `TextBox` control with the `TextMode` property set to `Password`.
5. Use a `ListBox` control, which allows multiple selections, and is best for long lists.
6. Set the `GroupName` property of each radio button to the same group.
7. A `Panel` control groups other controls together in one place, and enables you to make all the controls visible or invisible as a whole.
8. The `SelectedItem` property retrieves the `Text` property of the currently selected item of the control.
9. Set the control's `Visible` property to false. The control will be present, but won't be rendered until something changes the `Visible` property.
10. Set the `HyperLink` control's `Target` property to `_blank`.

Answers to Exercises

Exercise 2-1. This exercise isn't too tricky; you're just getting used to adding controls to the page, and seeing how nested `UpdatePanel` controls work. If the labels aren't updating independently of each other, make sure you have the buttons and labels inside the appropriate update panels. If not, you can drag them to their proper places. Also make sure, in the code-behind file, that your event handler is changing the text of the appropriate label. Example D-2 shows the markup for this exercise.

Example D-2. *The markup file for Exercise 2-1*

```
<%@ Page Language="VB" AutoEventWireup="true" CodeFile="Default.aspx.vb"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 2-1</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:ScriptManager ID="ScriptManager1" runat="server" />
        <div>
            Page Loaded at:
            <asp:Label ID="lblPageLoad" runat="server" Text="Label"
                 Width="200px"></asp:Label>
            <asp:Button ID="btnPostback" runat="server" Text="Postback" />
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    Partial-Page update at: <asp:Label ID="lblPartialUpdate"
                          runat="server" Text="Label"
                        Width="200px"></asp:Label>
```

Example D-2. *The markup file for Exercise 2-1 (continued)*

```
                    <asp:Button ID="btnPartialUpdate" runat="server"
                         OnClick="btnPartialUpdate_Click"
                        Text="Partial Update" />
                    <asp:UpdatePanel ID="UpdatePanel2" runat="server">
                        <ContentTemplate>
                            Another partial-page update at: <asp:Label
                                ID="lblOtherPartialUpdate" runat="server"
                                Text="Label" Width="200px"></asp:Label>
                            <asp:Button ID="btnOtherPartialUpdate" runat="server"
                                 OnClick="btnOtherPartialUpdate_Click"
                                Text="Another Partial Update" />
                        </ContentTemplate>
                    </asp:UpdatePanel>
                </ContentTemplate>
            </asp:UpdatePanel>
        </div>
    </form>
</body>
</html>
```

Example D-3 shows the code-behind file for this exercise.

Example D-3. *The code-behind file for Exercise 2-1*

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
      Handles Me.Load
        lblPageLoad.Text = DateTime.Now
    End Sub

    Protected Sub btnPartialUpdate_Click(ByVal sender As Object, ByVal e _
      As System.EventArgs)
        lblPartialUpdate.Text = DateTime.Now
    End Sub

    Protected Sub btnOtherPartialUpdate_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        lblOtherPartialUpdate.Text = DateTime.Now
    End Sub
End Class
```

Exercise 2-2. This exercise isn't too hard, and there are several valid solutions. The only challenge comes in choosing the best controls for the situation. The choice of ice cream type is a somewhat long list, and only allows for one selection, so a `DropDownList` control is probably best. For the toppings, again, it's a long list, but multiple selections are possible, so you could use a `ListBox`, but you need to make sure to set the `SelectionMode` property to `Multiple`. The choice of cone or dish is much simpler; there are only two options, and they're mutually exclusive, so a pair of `RadioButton` controls is the way to go here. Be sure to give them a common `GroupName`, so they'll be part of the same group.

Of course, that's not the only way to solve the problem. For an ice cream parlor, you might think it's good marketing to display all your flavors and toppings for your customers to choose from. In that case, you could use a `RadioButtonList` for the ice cream, and a `CheckBoxList` for the toppings. When you're designing a page, you need to consider all the customer's requirements.

One solution to this exercise is shown in Figure D-1. We didn't use much fancy styling here, but we did put the controls in a table to make the layout easier. The markup for this solution is shown in Example D-4.

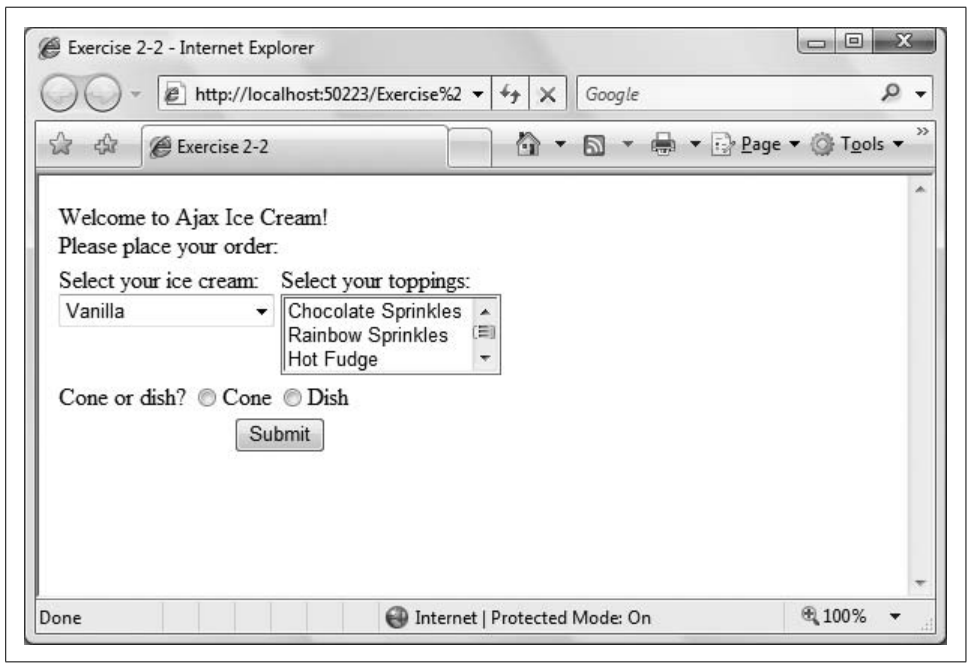


Figure D-1. *One solution to Exercise 2-2.*

Example D-4. *Markup for one solution to Exercise 2-2*

```
<%@ Page Language="VB" AutoEventWireup="true" CodeFile="Default.aspx.vb"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 2-2</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <table>
                <tr>
```

Example D-4. *Markup for one solution to Exercise 2-2 (continued)*

```
        <td colspan=2>
            Welcome to Ajax Ice Cream!<br />
            Please place your order:</td>
    </tr>
    <tr valign=top>
        <td>
            Select your ice cream:<br />
            <asp:DropDownList ID="ddlIceCream" runat="server">
                <asp:ListItem Value="Van">Vanilla</asp:ListItem>
                <asp:ListItem Value="Choc">Chocolate</asp:ListItem>
                <asp:ListItem Value="Straw">Strawberry</asp:ListItem>
                <asp:ListItem Value="Mint">Mint Chocolate
                                          Chip</asp:ListItem>
                <asp:ListItem Value="ButPec">Butter
                                            Pecan</asp:ListItem>
                <asp:ListItem Value="Coff">Coffee</asp:ListItem>
                <asp:ListItem Value="Pist">Pistachio</asp:ListItem>
                <asp:ListItem Value="Coco">Coconut</asp:ListItem>
                <asp:ListItem Value="Bub">Bubble Gum</asp:ListItem>
                <asp:ListItem Value="CotCan">Cotton
                                             Candy</asp:ListItem>
            </asp:DropDownList></td>
        <td>
            Select your toppings:<br />
            <asp:ListBox ID="lbToppings" runat="server" Rows="3"
                         SelectionMode="Multiple">
                <asp:ListItem Value="CSprink">Chocolate
                                              Sprinkles</asp:ListItem>
                <asp:ListItem Value="RSprink">Rainbow
                                              Sprinkles</asp:ListItem>
                <asp:ListItem Value="HFudge">Hot Fudge</asp:ListItem>
                <asp:ListItem Value="Carm">Caramel</asp:ListItem>
                <asp:ListItem Value="CDough">Cookie
                                             Dough</asp:ListItem>
                <asp:ListItem Value="Oreo">Oreo Cookies</asp:ListItem>
                <asp:ListItem Value="Pretz">Pretzel bits</asp:ListItem>
                <asp:ListItem Value="Nuts">Crushed
                                           Walnuts</asp:ListItem>
                <asp:ListItem Value="CBean">Coffee beans</asp:ListItem>
                <asp:ListItem Value="Candy">Crushed Candy
                                            Bars</asp:ListItem>
            </asp:ListBox></td>
    </tr>
    <tr>
        <td colspan=2>
            Cone or dish?
            <asp:RadioButton ID="rbCone" runat="server"
                             GroupName="grpConeDish" Text="Cone" />
            <asp:RadioButton ID="rbDish" runat="server"
                             GroupName="grpConeDish" Text="Dish" />
                             </td>
    </tr>
```

Example D-4. *Markup for one solution to Exercise 2-2 (continued)*

```
                <tr>
                    <td colspan=2 align=center>
                        <asp:Button ID="btnSubmit" runat="server" Text="Submit"
                           /></td>
                </tr>
            </table>
        </div>
    </form>
</body>
</html>
```

Exercise 2-3. Creating the web page for this exercise is simple; you just need a `TextBox`, a `Label`, a `Button`, and some plain text to tell the user what to do. The only hitch is that you need to remember to set the `TextMode` property of the `TextBox` to `Password`. The next step is writing the event handler for the Submit button. All you want to do is set the `Text` property of the `Label` to be the same as the `Text` property of the `TextBox`. For that, you need just one line:

```
lblPassword.Text = txtPassword.Text
```

Notice that even though the user can't see the text that's typed in the `TextBox`, the page can, and can assign that value to another control (the label, in this case). This is, of course, a terrible security practice, but it illustrates the point. Example D-5 shows the markup file for this exercise, and Example D-6 shows the short event hander for the Submit button.

Example D-5. *The markup for Exercise 2-3*

```
<%@ Page Language="VB" AutoEventWireup="true" CodeFile="Default.aspx.vb"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 2-3</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            Enter your password:
            <asp:TextBox ID="txtPassword" runat="server"
                          TextMode="Password"></asp:TextBox><br />
            The password you entered is:
            <asp:Label ID="lblPassword" runat="server"></asp:Label>
            <br />
            <asp:Button ID="btnSubmit" runat="server" Text="Submit" /></div>
    </form>
</body>
</html>
```

Example D-6. *The event handler for the Submit button in Exercise 2-3*

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub btnSubmit_Click(ByVal sender As Object, ByVal e As _
                                  System.EventArgs) Handles btnSubmit.Click
        lblPassword.Text = txtPassword.Text
    End Sub
End Class
```

Exercise 2-4. In this exercise, you're combining the assignment you did in Exercise 2-3 with the `DropDownList` control you learned about earlier. When you create the drop-down list, either with the ListItem editor, or by hand, you assign the ISBN number to the `Value` property of the `ListItem`. That way, when the user makes a selection, the user never sees the ISBN number, but it appears in the output anyway. You can see how this would be useful for a database, so that you can hide your internal system from the users, but still allow them to choose by title.

The only twist for this exercise is that it lacks a Submit button. Therefore, to get the postback, you can set the `DropDownList` control's `AutoPostBack` property to `true`, so that the page posts back every time the user makes a selection. That means you have to put the event handler in the `Load` event for the page, just as you did in the Postbacks example in this chapter. (You could also use a `SelectedItemChanged` handler, if you prefer.) The event handler works a bit differently than in Exercise 2-3:

```
lblID.Text = ddlBookList.SelectedValue
lblTitle.Text = ddlBookList.SelectedItem.Text
```

To assign the text to the ID label, you need to retrieve the `Value` property of the selected item in the drop-down list—that's the `SelectedValue` property. Assigning the title text is slightly trickier: you retrieve the `Text` property of the `SelectedItem` property of the drop-down list: `ddlBookList.SelectedItem.Text`.

The markup for this exercise is shown in Example D-7, and the event handler is in Example D-8.

Example D-7. *The markup for Exercise 2-4*

```
<%@ Page Language="VB" AutoEventWireup="true" CodeFile="Default.aspx.vb" Inherits="_
Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 2-4</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            Which book are you interested in?<br />
```

Example D-7. *The markup for Exercise 2-4 (continued)*

```
            <asp:DropDownList ID="ddlBookList" runat="server" AutoPostBack="true">
                <asp:ListItem Value="00916X">Programming ASP.NET</asp:ListItem>
                <asp:ListItem Value="006993">Programming C#</asp:ListItem>
                <asp:ListItem Value="004385">Programming Visual Basic
                                              .NET</asp:ListItem>
                <asp:ListItem Value="102097">Learning C# 2005</asp:ListItem>
            </asp:DropDownList><br />
                Thank you for your interest in <br />
                <asp:Label ID="lblID" runat="server" Text="ID number"></asp:Label>
            <br />
                <asp:Label ID="lblTitle" runat="server"
                             Text="Title"></asp:Label></div>
    </form>
</body>
</html>
```

Example D-8. *The event handler for Exercise 2-4*

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
                           Handles Me.Load
        lblID.Text = ddlBookList.SelectedValue
        lblTitle.Text = ddlBookList.SelectedItem.Text
    End Sub
End Class
```

Chapter 3: Snappier Web Sites with AJAX

Answers to Quiz Questions

1. To use AJAX controls on your web site, you need to add a `ScriptManager` control to your page. ASP.NET takes care of the rest of the details.
2. The `EnablePartialRendering` property is the critical property of the `ScriptManager` control, which is why it is set to `True` by default.
3. The `UpdatePanel` control is the key control that enables asynchronous updates.
4. You need to handle the `Navigate` event of the `ScriptManager` control to modify the behavior of the browser Forward and Back buttons for AJAX controls. You also have to make sure the `EnableHistory` property of the `ScriptManager` control is set to `True`.
5. No, you can't. The extender controls need to have a target control to extend; they don't work alone.
6. All the Extender controls have a property called `TargetControlID`, which indicates the control that the extender acts on.

7. You have to set the WatermarkText property from Source view; it's not visible in Design view.
8. It allows you to hide choices for a control within an UpdatePanel, saving valuable screen space.
9. The Commit( ) method is the method of the PopUpControlExtender that causes the target control to display the results.
10. A CollapsiblePanelExtender, logically enough, lets you expand or collapse a Panel control.

Answers to Exercises

Exercise 3-1. Start by creating a web site called Exercise 3-1. When the new web site opens, switch to Design View, and drag a ScriptManager control onto the page, or else you won't get very far. Next, type in "Shipping State:" and add a TextBox control called txtState. Set its Text property to "Click Here," its ReadOnly property to True, and its Width property to 70px. Those are the controls the user will see to start out. Now, add a standard Panel control below the TextBox, and name it pnlPopup. Inside the Panel, place an UpdatePanel from the AJAX Extensions section of the Toolbox. Inside the UpdatePanel, place a DropDownList control, and call it ddlStates. Set the AutoPostBack property of ddlStates to True—this is an important step. Now use the ListItem Editor to fill in the six items. Be sure to set the Text of each ListItem to the full name of the state, and the Value to the state's postal code.

So far, it's easy. Now, click on the TextBox, open its Smart Tag, and select Add Extender. When the Extender Wizard opens, select PopupControlExtender. You can leave the default name. Switch to Source view to set the extender's properties. The Panel is the control that you want to pop up, so set PopupControlID to pnlPopup. Set Position to Bottom, just to make it neater.

The PopupControlExtender is in place now, and ready to go, but it still needs to know what to do, and for that, you need an event handler for the DropDownList. Double-click ddlStates in Design view to open the default event handler, SelectedIndexChanged. You need to call the Commit method of pceStates here, and pass in the value that the user has selected, not the text. So, add this line of code to the event handler:

```
Protected Sub ddlStates_SelectedIndexChanged(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles ddlStates.SelectedIndexChanged
   txtState_PopupControlExtender.Commit(ddlStates.SelectedValue)
End Sub
```

Run your application. You should find that the panel is hidden until you click it, and that the value is automatically passed to txtState when you make a selection. Example D-9 shows the markup for *Default.aspx* for this exercise.

Example D-9. *The markup file for Exercise 3-1*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>
<%@ Register assembly="AjaxControlToolkit" namespace="AjaxControlToolkit"
tagprefix="cc1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 3-1</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>

        Shipping State:
        <asp:TextBox ID="txtState" runat="server" ReadOnly="True" Width="70px">Click Here
        </asp:TextBox>
        <cc1:PopupControlExtender ID="txtState_PopupControlExtender" runat="server"
            DynamicServicePath="" Enabled="True" ExtenderControlID=""
            TargetControlID="txtState" PopupControlID="pnlPopup"
            Position="Bottom">
        </cc1:PopupControlExtender>

        <asp:Panel ID="pnlPopup" runat="server">
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:DropDownList ID="ddlStates" runat="server" AutoPostBack="True">
                        <asp:ListItem Value="CT">Connecticut</asp:ListItem>
                        <asp:ListItem Value="MA">Massachusetts</asp:ListItem>
                        <asp:ListItem Value="NJ">New Jersey</asp:ListItem>
                        <asp:ListItem Value="NY">New York</asp:ListItem>
                        <asp:ListItem Value="PA">Pennsylvania</asp:ListItem>
                        <asp:ListItem Value="RI">Rhode Island</asp:ListItem>
                    </asp:DropDownList>
                </ContentTemplate>
            </asp:UpdatePanel>
        </asp:Panel>
    </div>

    </form>
</body>
</html>
```

Example D-10 shows the very brief code-behind file.

Example D-10. *The code-behind file for Exercise 3-1*

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub ddlStates_SelectedIndexChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles ddlStates.SelectedIndexChanged
        txtState_PopupControlExtender.Commit(ddlStates.SelectedValue)
    End Sub
End Class
```

Exercise 3-2. This is a fairly simple extender to work with—it does what it says it does. To start, create a new web site, and call it Exercise 3-2. First add the `ScriptManager` control to enable AJAX. Then, add the `Panel` control, call it `pnlRounded`, and set its `Width` to 150, its `Height` to 100, and its `BackColor` to `LightGray` (or whatever you prefer). Add the `Label` control inside the `Panel`, call it `lblRounded`, and set its `Width` to 50, its `BackColor` to `DarkGray`, its `ForeColor` to `White`, and its `Font-Bold` to `True`.

So far, nothing particularly interesting has happened. Select `pnlRounded`, click the Smart Tag, and select `RoundedCornersExtender`. Then do the same for `lblRounded`. For the rest, you'll have to check out the online documentation, which will tell you that there's a property called `Radius`, and one called `Corners`. Those properties aren't available in Design view, so switch to Source view and set the `Radius` for `lblRounded_RoundedCornersExtender` to 2; for `pnlRounded_RoundedCornersExtender`, set the `Radius` to 8, and the `Corners` to `Top`, so that the bottom corners of the `Panel` will be left square. There you go! It's not the most exciting extender, but you can see how you might use it to add a little bit of style to your forms. The markup for this page is in Example D-11. There is no code-behind file for this example.

Example D-11. *The markup file for Exercise 3-2*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>
<%@ Register assembly="AjaxControlToolkit" namespace="AjaxControlToolkit"
tagprefix="cc1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 3-2</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
```

Example D-11. *The markup file for Exercise 3-2 (continued)*

```
        <asp:Panel ID="pnlRounded" runat="server" Height="100px"
                   Width="150px" BackColor="LightGray">
            <br />
             
            <asp:Label ID="lblRounded" runat="server" Text="Label"
                Width="50px" BackColor="DarkGray" ForeColor="White"
                Font-Bold="true"></asp:Label>
            <cc1:RoundedCornersExtender ID="lblRounded_RoundedCornersExtender"
                runat="server" Enabled="True" TargetControlID="lblRounded"
                Radius=2>
            </cc1:RoundedCornersExtender>
        </asp:Panel>

        <cc1:RoundedCornersExtender ID="pnlRounded_RoundedCornersExtender"
            runat="server" Enabled="True" TargetControlID="pnlRounded"
            Radius=8 Corners="Top">
        </cc1:RoundedCornersExtender>

    </div>
    </form>
</body>
</html>
```

If you experiment with this extender, you'll find that it doesn't work on other controls, such as the `TextBox` or the `DropDownList`. Sometimes, the documentation doesn't tell you everything you need to know, and you can only find out by trial and error.

Exercise 3-3. This exercise is slightly tricky because of the need for two textboxes. Start by creating an page named Exercise 3-3, and add the required `ScriptManager` control. Type "Volume Level:" and then add a `TextBox` control. This is the `TextBox` that you want to see in the finished page, so name it `txtVolume_Bound`. Set its `Width` to 15px.

That's the easy part. Now, add another `TextBox`, called `txtVolume`, on a new line. None of the display properties matter on this `TextBox` because users will never see it. Click on the Smart Tag, select the SliderExtender, and accept the default name. Switch to Source view and set the extender's `Maximum` property to 10, and the `BoundControlID` to `txtVolume_Bound`. Run the application, and the slider should work as you'd expect, although you'll only see one `TextBox`. Also, notice that if you type a number in the text box, the slider's handle moves to match. The markup for this page is shown in Example D-12. There is no code-behind file.

Example D-12. *The markup for Exercise 3-3*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>
<%@ Register assembly="AjaxControlToolkit" namespace="AjaxControlToolkit"
tagprefix="cc1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

Example D-12. *The markup for Exercise 3-3 (continued)*

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 3-3</title>
</head>
<body>
    <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <div>

        Volume Level:
        <asp:TextBox ID="txtVolume_Bound" runat="server" Width="15px">
        </asp:TextBox>
        <br />
        <asp:TextBox ID="txtVolume" runat="server"></asp:TextBox>
        <cc1:SliderExtender ID="txtVolume_SliderExtender" runat="server"
            Enabled="True" TargetControlID="txtVolume" Maximum="10"
            BoundControlID="txtVolume_Bound">
        </cc1:SliderExtender>

    </div>
    </form>
</body>
</html>
```

As you can imagine, there are many possible uses for a control like this, and for many of them, you would actually want to hide the textbox the control extends. You could use it as an actual volume control, for one thing, or you could use several of them to make a color slider like you find in a drawing application.

Exercise 3-4. Creating the web site for this exercise isn't difficult. You need to make sure you start with a `ScriptManager` control and an `UpdatePanel` control. Then, place the `TextBox` and the `Button` controls inside the `UpdatePanel` and name them appropriately. Set the `EnableHistory` property of the `ScriptManager` control to `True` to enable the browser history for the AJAX controls on the page, which in this exercise is just the `TextBox` and the `Button` in the `UpdatePanel`.

Now you need to set a history point every time the user clicks the `Button`, which means you'll have to handle the `Click` event for the button. To do that, you'll need to define two variables, both as strings: `strHistoryText`, to hold the text that was in the `TextBox` when the Button was clicked, and `strTitle`, which isn't strictly necessary, but provides the text for the browser history list. In this case, we're just using the word "History:" followed by the same text that was in the `TextBox`. Then you call the `AddHistoryPoint()` method of the `ScriptManager`, passing in the string "TextBox" as the key, and then the two variables you created:

```
    Dim strHistoryText As String = txtHistory.Text
    Dim strTitle As String = "History: " + strHistoryText

    ScriptManager1.AddHistoryPoint("TextBox", strHistoryText, strTitle)
```

Now that the history points are enabled, you need to write an event handler for the `Navigate` event of the `ScriptManager` control, so that you can retrieve the values later. First, you test to see whether the value associated with the key "TextBox" is `Nothing` (that is, if there's nothing stored for that key). If it isn't, then you can set the value of the `TextBox` to the value you stored in that key:

```
    If Not e.State("TextBox") Is Nothing Then
        txtHistory.Text = e.State("TextBox")
    End If
```

The markup for this page is shown in Example D-13. The code-behind file is shown in Example D-14.

Example D-13. *The markup for Exercise 3-4*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 3-4</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:ScriptManager ID="ScriptManager1" runat="server" EnableHistory="True">
        </asp:ScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <asp:TextBox ID="txtHistory" runat="server"></asp:TextBox>
                <asp:Button ID="btnHistory" runat="server" Text="Save History" />
            </ContentTemplate>
        </asp:UpdatePanel>

    </div>
    </form>
</body>
</html>
```

Example D-14. *The code-behind file for Exercise 3-4*

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub btnHistory_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
                                Handles btnHistory.Click
        Dim strHistoryText As String = txtHistory.Text
        Dim strTitle As String = "History: " + strHistoryText

        ScriptManager1.AddHistoryPoint("TextBox", strHistoryText, strTitle)
    End Sub

    Protected Sub ScriptManager1_Navigate(ByVal sender As Object, _
                                        ByVal e As System.Web.UI.HistoryEventArgs) _
                                        Handles ScriptManager1.Navigate
        If Not e.State("TextBox") Is Nothing Then
            txtHistory.Text = e.State("TextBox")
        End If

    End Sub
End Class
```

Chapter 4: Saving and Retrieving Data

Answers to Quiz Questions

1. You need a `DataSource` control to provide a database connection to one of your other controls.
2. Binding is the process that allows controls to retrieve data from retrieved tables.
3. A connection string is a string that contains the information necessary to connect to a database on a server. You can store the connection string in the *web.config* file for later use.
4. Create (add a new record), Retrieve, Update (edit), and Delete.
5. Use the `GridView`'s Smart Tag and the "Choose Data Source" drop-down list.
6. Turn on paging by clicking the Smart Tag and selecting Enable Paging.
7. Enable Updating and Deleting from the Smart Tag.
8. Create an event handler for the `RowDataBound` event.
9. To allow users to edit data, the `LinqDataSource` must retrieve all the columns from the table. Therefore, you need to select the asterisk field (*) in the configure data source wizard.
10. In terms of Dynamic Data, there are two types of templates: field templates define how individual data types are displayed; page templates define the behavior of an entire set of data.

Answers to Exercises

Exercise 4-1. Here's one way to get a page that looks like Figure 4-42:

1. Create a new web site as usual. Name it Exercise 4-1.
2. Drag a `SqlDataSource` control onto the Design view.
3. Drag a `GridView` control onto the Design view. Click the `GridView`'s Smart Tag, go to the "Choose Data Source" drop-down list, and select `SqlDataSource1`.
4. Click "Configure Data Source" in the Smart Tag. The Configure Data Source Wizard starts. You can use the connection string from the other exercises in this chapter. Click Next.
5. On the next page, click the radio button marked "Specify columns from a table or view," and select the `Product` table from the drop-down list. Check the boxes for `ProductID`, `Name`, `ProductNumber`, `Color`, and `ListPrice`. The Wizard should look like Figure D-2.

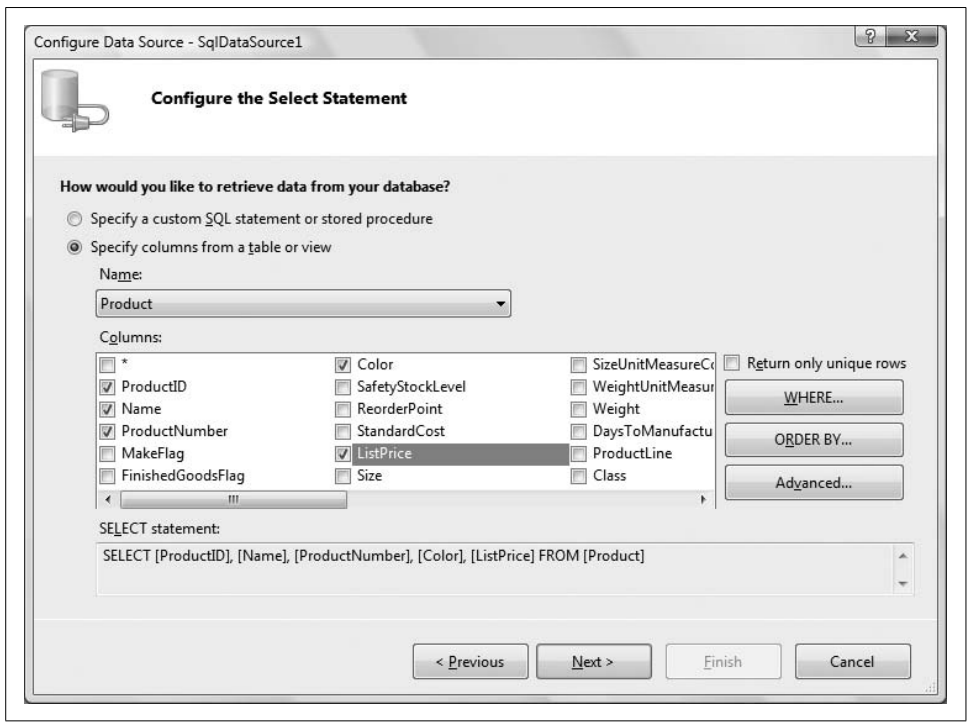


Figure D-2. *Creating the Select statement for your GridView.*

6. Click the WHERE button. In the Add WHERE Clause dialog, select Weight in the Column drop-down list, select > (greater than) in the Operator drop-down list, and None in the Source drop-down list. The Value field appears on the right side of the box; enter 100 in the field. The dialog should look like Figure D-3. Click Add to add the `WHERE` clause, and OK to return to the Wizard.

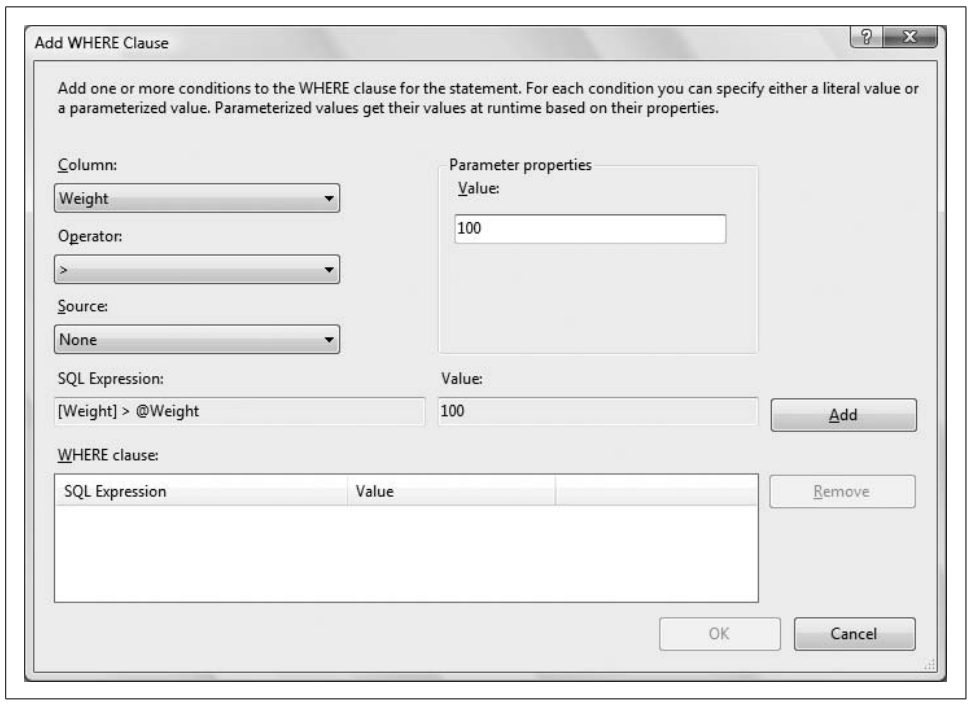


Figure D-3. *Adding the WHERE clause for your GridView.*

7. Click the Advanced button and check Generate INSERT, UPDATE, and DELETE statements and Use optimistic concurrency.
8. Click Next to go to the next page; then, click Finish to close the Wizard. Remember that this query won't work properly right now, so switch to Source view, find the four instances of `[Product]`, and change them to `[Production].[Product]`, like you did earlier in the chapter.
9. Return to Design View, click the Smart Tag for the `GridView`, and check the boxes for Enable Paging, Enable Sorting, Enable Editing, and Enable Deletion. Click AutoFormat, and select the Professional color scheme. Run your application.

The Source code for this application looks like Example D-15.

Example D-15. *The markup for Exercise 4-1*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 4-1</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString %>"

            SelectCommand="SELECT [ProductID], [Name], [ProductNumber], [Color],
                           [ListPrice] FROM [Production].[Product]
                           WHERE ([Weight] &gt; @Weight)"
            ConflictDetection="CompareAllValues"
            DeleteCommand="DELETE FROM [Production].[Product]
                           WHERE [ProductID] = @original_ProductID
                           AND [Name] = @original_Name
                           AND [ProductNumber] = @original_ProductNumber
                           AND [Color] = @original_Color
                           AND [ListPrice] = @original_ListPrice"
            InsertCommand="INSERT INTO [Production].[Product] ([Name],
                           [ProductNumber], [Color], [ListPrice])
                           VALUES (@Name, @ProductNumber, @Color, @ListPrice)"
            OldValuesParameterFormatString="original_{0}"
            UpdateCommand="UPDATE [Production].[Product] SET
                           [Name] = @Name, [ProductNumber] = @ProductNumber,
                           [Color] = @Color, [ListPrice] = @ListPrice
                           WHERE [ProductID] = @original_ProductID
                           AND [Name] = @original_Name
                           AND [ProductNumber] = @original_ProductNumber
                           AND [Color] = @original_Color
                           AND [ListPrice] = @original_ListPrice">
            <SelectParameters>
                <asp:Parameter DefaultValue="100" Name="Weight" Type="Decimal" />
            </SelectParameters>
            <DeleteParameters>
                <asp:Parameter Name="original_ProductID" Type="Int32" />
                <asp:Parameter Name="original_Name" Type="String" />
                <asp:Parameter Name="original_ProductNumber" Type="String" />
                <asp:Parameter Name="original_Color" Type="String" />
                <asp:Parameter Name="original_ListPrice" Type="Decimal" />
            </DeleteParameters>
            <UpdateParameters>
                <asp:Parameter Name="Name" Type="String" />
                <asp:Parameter Name="ProductNumber" Type="String" />
                <asp:Parameter Name="Color" Type="String" />
```

Example D-15. *The markup for Exercise 4-1 (continued)*

```
                <asp:Parameter Name="ListPrice" Type="Decimal" />
                <asp:Parameter Name="original_ProductID" Type="Int32" />
                <asp:Parameter Name="original_Name" Type="String" />
                <asp:Parameter Name="original_ProductNumber" Type="String" />
                <asp:Parameter Name="original_Color" Type="String" />
                <asp:Parameter Name="original_ListPrice" Type="Decimal" />
            </UpdateParameters>
            <InsertParameters>
                <asp:Parameter Name="Name" Type="String" />
                <asp:Parameter Name="ProductNumber" Type="String" />
                <asp:Parameter Name="Color" Type="String" />
                <asp:Parameter Name="ListPrice" Type="Decimal" />
            </InsertParameters>
        </asp:SqlDataSource>
        <asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1"
            AllowPaging="True" AllowSorting="True" CellPadding="4" ForeColor="#333333"
            GridLines="None">
            <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
            <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
            <Columns>
                <asp:CommandField ShowDeleteButton="True" ShowEditButton="True" />
            </Columns>
            <PagerStyle BackColor="#284775" ForeColor="White" HorizontalAlign="Center" />
            <SelectedRowStyle BackColor="#E2DED6" Font-Bold="True" ForeColor="#333333" />
            <HeaderStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
            <EditRowStyle BackColor="#999999" />
            <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
        </asp:GridView>

    </div>
    </form>
</body>
</html>
```

Exercise 4-2. Adding labels and textboxes to your page is easy. The tricky bit is that you need to change the content of the textboxes whenever the user selects a row in the `GridView`. Fortunately, you've seen the `SelectedIndexChanged` event, which makes it easy to send data to controls based on a user's selections in the `GridView`.

Start by copying the web site from Exercise 4-1 to a new web site, Exercise 4-2, as explained in Appendix B.

Click the `GridView`'s Smart Tag, and check the Enable Selection checkbox. The Select links show up next to the Edit and Delete links that are already there.

Drag a Label onto the Design view, below the `GridView`. Name it `lblName`, and change the text to "Name:". Drag a `TextBox` next to the Label, name it `txtName`, and set its `ReadOnly` property to `True`. Add another `Label`, named `lblColor`, with text of "Color:". Finally, add a second `TextBox`, name it `txtColor`, and set its `ReadOnly` property to `True`.

You're got all the parts in place, but now you need to connect the Select link in the GridView to the two TextBoxes. Double-click one of the Select links to open the SelectedIndexChanged event handler for GridView1. Add the following highlighted code:

```
Protected Sub GridView1_SelectedIndexChanged(ByVal sender As Object,_
  ByVal e As System.EventArgs) Handles GridView1.SelectedIndexChanged
  If GridView1.SelectedRow.RowType = DataControlRowType.DataRow Then
    Dim cellName As TableCell = GridView1.SelectedRow.Cells(2) ' Name column
    Dim cellColor As TableCell = GridView1.SelectedRow.Cells(4) ' Color column
    txtName.Text = cellName.Text
    txtColor.Text = cellColor.Text
  End If
End Sub
```

This is very similar to what you did earlier in the chapter. GridView1.SelectedRow represents the currently selected row—the one the user clicked. First, check to see if SelectedRow is a DataRow. If it is, you create a cell variable called cellName, and set that equal to the cell at index 2 in SelectedRow, because you know that's the Name column (remember that the row starts at index 0). Similarly, you create a cell cellColor, which is set to the cell at index 4. Then, you assign the Text properties of each of these cells to the respective TextBoxes.

Run your application and try it out. As you select different rows, the values in the TextBoxes change. You can see how this would be useful for customers to select products on your order form.

Example D-16 shows the source code for Exercise 4-2.

Example D-16. *The markup for Exercise 4-2*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
 Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 4-2</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString %>"

            SelectCommand="SELECT [ProductID], [Name], [ProductNumber], [Color],
                        [ListPrice] FROM [Production].[Product]
                        WHERE ([Weight] &gt; @Weight)"
            ConflictDetection="CompareAllValues"
```

Example D-16. *The markup for Exercise 4-2 (continued)*

```
        DeleteCommand="DELETE FROM [Production].[Product]
                       WHERE [ProductID] = @original_ProductID
                       AND [Name] = @original_Name
                       AND [ProductNumber] = @original_ProductNumber
                       AND [Color] = @original_Color
                       AND [ListPrice] = @original_ListPrice"
        InsertCommand="INSERT INTO [Production].[Product] ([Name],
                       [ProductNumber], [Color], [ListPrice])
                       VALUES (@Name, @ProductNumber, @Color, @ListPrice)"
        OldValuesParameterFormatString="original_{0}"
        UpdateCommand="UPDATE [Production].[Product] SET
                       [Name] = @Name, [ProductNumber] = @ProductNumber,
                       [Color] = @Color, [ListPrice] = @ListPrice
                       WHERE [ProductID] = @original_ProductID
                       AND [Name] = @original_Name
                       AND [ProductNumber] = @original_ProductNumber
                       AND [Color] = @original_Color
                       AND [ListPrice] = @original_ListPrice">
        <SelectParameters>
            <asp:Parameter DefaultValue="100" Name="Weight" Type="Decimal" />
        </SelectParameters>
        <DeleteParameters>
            <asp:Parameter Name="original_ProductID" Type="Int32" />
            <asp:Parameter Name="original_Name" Type="String" />
            <asp:Parameter Name="original_ProductNumber" Type="String" />
            <asp:Parameter Name="original_Color" Type="String" />
            <asp:Parameter Name="original_ListPrice" Type="Decimal" />
        </DeleteParameters>
        <UpdateParameters>
            <asp:Parameter Name="Name" Type="String" />
            <asp:Parameter Name="ProductNumber" Type="String" />
            <asp:Parameter Name="Color" Type="String" />
            <asp:Parameter Name="ListPrice" Type="Decimal" />
            <asp:Parameter Name="original_ProductID" Type="Int32" />
            <asp:Parameter Name="original_Name" Type="String" />
            <asp:Parameter Name="original_ProductNumber" Type="String" />
            <asp:Parameter Name="original_Color" Type="String" />
            <asp:Parameter Name="original_ListPrice" Type="Decimal" />
        </UpdateParameters>
        <InsertParameters>
            <asp:Parameter Name="Name" Type="String" />
            <asp:Parameter Name="ProductNumber" Type="String" />
            <asp:Parameter Name="Color" Type="String" />
            <asp:Parameter Name="ListPrice" Type="Decimal" />
        </InsertParameters>
    </asp:SqlDataSource>
    <asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1"
        AllowPaging="True" AllowSorting="True" CellPadding="4" ForeColor="#333333"
        GridLines="None">
        <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
        <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
        <Columns>
```

Example D-16. *The markup for Exercise 4-2 (continued)*

```
                <asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
                    ShowSelectButton="True" />
            </Columns>
            <PagerStyle BackColor="#284775" ForeColor="White" HorizontalAlign="Center" />
            <SelectedRowStyle BackColor="#E2DED6" Font-Bold="True" ForeColor="#333333" />
            <HeaderStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
            <EditRowStyle BackColor="#999999" />
            <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
        </asp:GridView>

        <asp:Label ID="lblName" runat="server" Text="Name:"></asp:Label>
        <asp:TextBox ID="txtName" runat="server" ReadOnly="True"></asp:TextBox>
        <asp:Label ID="lblColor" runat="server" Text="Color: "></asp:Label>
        <asp:TextBox ID="txtColor" runat="server" ReadOnly="True"></asp:TextBox>

    </div>
    </form>
</body>
</html>
```

The code-behind file is shown in Example D-17.

Example D-17. *The code-behind file for Exercise 4-2*

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub GridView1_SelectedIndexChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles GridView1.SelectedIndexChanged

        If GridView1.SelectedRow.RowType = DataControlRowType.DataRow Then
            Dim cellName As TableCell = GridView1.SelectedRow.Cells(2) ' Name column
            Dim cellColor As TableCell = GridView1.SelectedRow.Cells(4) ' Color column
            txtName.Text = cellName.Text
            txtColor.Text = cellColor.Text
        End If

    End Sub
End Class
```

Exercise 4-3. This is mostly a simple exercise, but it shows you how the AJAX Update panels interact with `GridViews`.

1. Create a new web site called Exercise 4-3.
2. Drag a `ScriptManager` control onto the page first to enable AJAX for this site. Next, drag an `UpdatePanel` onto your page. Everything else goes inside the `UpdatePanel`.
3. Add some text inside the `UpdatePanel`, "Select the table you would like to see:". Drag two radio buttons into the `UpdatePanel` beneath that text. Name the first one `rbEmployee`, set its text to "Show Employees," and set its `GroupName` to `grpEmployeeCustomer`. Name the second radio button `rbCustomer`, set its text to

"Show Customers," and set its GroupName to grpEmployeeCustomer as well. Now both buttons are part of the same group.

4. Add a SQLDataSource to the page, and name it sdsEmployee. Add a GridView to the page, and call it gvEmployee.
5. Time to give gvEmployee something to do. Click its Smart Tag, select sdsEmployee as the data source, and configure sdsEmployee to retrieve the EmployeeID, ManagerID, and Title columns from the Employee table. Remember to go to Source view and add the schema. In this case, you need to replace [Employee] with [HumanResources].[Employee]. Enable paging and sorting on gvEmployee, and give it whatever formatting you like.
6. Now drag another SQLDataSource (sdsCustomer) and another GridView (gvCustomer) onto the page.
7. gvCustomer needs some data as well. Click its Smart Tag, select sdsCustomer as the data source, and configure sdsCustomer to retrieve the CustomerID, AccountNumber, and CustomerType fields from the Customer table. Again, go to source view and change [Customer] to [Sales].[Customer]. Enable paging and sorting, and apply whatever format you want.
8. Everything's easy so far, right? Now for the only tricky bit. You want gvEmployee to be visible when the rbEmployee radio button is clicked; when rbCustomer is clicked, only gvCustomer should be visible. You've seen how to do that in Chapter 2.
9. Start out with neither GridView visible. Click each GridView, and set its Visible property to false.
10. Double-click rbEmployee, which creates the rbEmployee_CheckChanged event handler for you. You want to change the visibility of gvEmployee to be the same as the Checked value of rbEmployee, and you want gvCustomer.Visible to have the same value as rbCustomer.Checked. Because you know that rbEmployee and rbCustomer can't both be checked at the same time (because they're part of the same radio button group), you know that only one GridView will be visible at a time. Here's the code to add to rbEmployee_CheckChanged:

```
Protected Sub rbEmployee_CheckedChanged(ByVal sender As Object, _
ByVal e As System.EventArgs)
   gvEmployee.Visible = rbEmployee.Checked
   gvCustomer.Visible = rbCustomer.Checked
End Sub
```

11. Go back to the *.aspx file*, click rbEmployee, and set its AutoPostBack property to true, so that the table will show as soon as the button is clicked.
12. Now double-click rbCustomer to create the rbCustomer_CheckChanged event handler, and add the exact same code. Also make sure to set rbCustomer's AutoPostBack property to true.

13. Test out your application. As you click each radio button, the table displayed should change. You will see some delay the first time you click the radio button, because the data for the `GridView` is being sent to the client, but after that, there's no postback, even if you change the page or sort the `GridView`.

Example D-18 shows the markup file for Exercise 4-3.

Example D-18. *The markup file for Exercise 4-3*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 4-3</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                Select the table you would like to see:
                <br />
                <asp:RadioButton ID="rbEmployee" runat="server"
                    GroupName="grpEmployeeCustomer"
                    Text="Show Employees" AutoPostBack="True" />
                <br />
                <asp:RadioButton ID="rbCustomer" runat="server"
                    GroupName="grpEmployeeCustomer"
                    Text="Show Customers" AutoPostBack="True" />
                <asp:SqlDataSource ID="sdsEmployee" runat="server"
                    ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString
                    %>"
                    SelectCommand="SELECT [EmployeeID], [ManagerID], [Title]
                                   FROM [HumanResources].[Employee]">
                </asp:SqlDataSource>
                <asp:GridView ID="gvEmployee" runat="server" DataSourceID="sdsEmployee"
                    CellPadding="4" ForeColor="#333333" GridLines="None"
                    AllowPaging="True"
                    AllowSorting="True" Visible="False">
                    <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
                    <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
                    <PagerStyle BackColor="#284775" ForeColor="White"
                    HorizontalAlign="Center" />
                    <SelectedRowStyle BackColor="#E2DED6" Font-Bold="True"
                    ForeColor="#333333" />
                    <HeaderStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
                    <EditRowStyle BackColor="#999999" />
```

Example D-18. *The markup file for Exercise 4-3 (continued)*

```
                <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
            </asp:GridView>
            <asp:SqlDataSource ID="sdCustomer" runat="server"
                ConnectionString="<%$ ConnectionStrings:AdventureWorksConnectionString
                %>"
                SelectCommand="SELECT [CustomerID], [AccountNumber], [CustomerType]
                               FROM [Sales].[Customer]">
            </asp:SqlDataSource>
            <asp:GridView ID="gvCustomer" runat="server" DataSourceID="sdCustomer"
                AllowPaging="True" AllowSorting="True" CellPadding="4"
                ForeColor="#333333"
                GridLines="None" Visible="False">
                <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
                <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
                <PagerStyle BackColor="#284775" ForeColor="White"
                HorizontalAlign="Center" />
                <SelectedRowStyle BackColor="#E2DED6" Font-Bold="True"
                ForeColor="#333333" />
                <HeaderStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
                <EditRowStyle BackColor="#999999" />
                <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
            </asp:GridView>
        </ContentTemplate>
    </asp:UpdatePanel>

    </div>
    </form>
</body>
</html>
```

Example D-19 shows the code-behind file for Exercise 4-3.

Example D-19. *The code-behind file for Exercise 4-3*

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub rbEmployee_CheckedChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles rbEmployee.CheckedChanged
        gvEmployee.Visible = rbEmployee.Checked
        gvCustomer.Visible = rbCustomer.Checked

    End Sub

    Protected Sub rbCustomer_CheckedChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles rbCustomer.CheckedChanged
        gvEmployee.Visible = rbEmployee.Checked
        gvCustomer.Visible = rbCustomer.Checked

    End Sub
End Class
```

Exercise 4-4. You're going to need several `DataSource` controls for this exercise—one for each of the drop-down lists, and another for the `GridView`.

1. Add a `SqlDataSource`, a `Label`, and a `DropDownList` to your page. Call the data source `sdsCategorySource`, the label `lblCategory`, and the drop-down list `ddlCategory`. Configure `sdsCategorySource` to retrieve the `ProductCategoryID` and `Name` columns from the `ProductCategory` table (remember to go to Source view and add the `[Production]` schema).
2. Click the Smart Tag on `ddlCategory` and select "Choose Data Source." For the "Select a data source" field, choose `sdsCategorySource`. For "Select a data field to display in the DropDownList," select the Name column—this is what you want the user to see in the drop-down list. For "Select a data field for the value of the DropDownList," select `ProductCategoryID`. The control will automatically associate each name with its appropriate value within the control, so you can use them later. The Choose Data Source page should look like Figure D-4. If you don't see anything in the drop-down lists, click the "Refresh Schema" link at the bottom of the dialog.

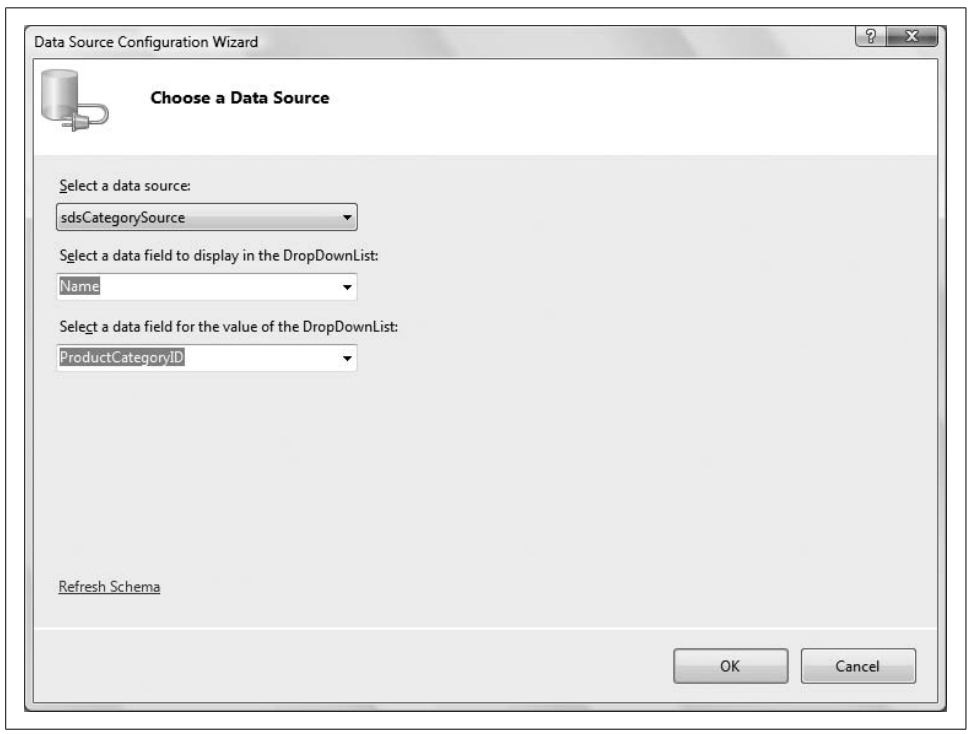


Figure D-4. *Select the source, name, and value properties for ddlCategory.*

3. That's one control set up. Add some text to the label, and test the application now to make sure everything is working. Your drop-down list won't do much of anything yet, but if the product categories appear in the list, you'll know you configured your data source properly.

For the Subcategory drop-down list, you want to only show the subcategories that are connected to the category the user chose. That's only slightly trickier—you'll need a `WHERE` clause.

1. Add a `SqlDataSource` (and call it `sdsSubcategorySource`), a `DropDownList` (called `ddlSubcategory`), and also a label for the drop-down list. Then, configure `sdsSubcategorySource` to retrieve the `ProductSubcategoryID` and `Name` columns from the `ProductSubcategory` table.
2. Click the WHERE button to add a `WHERE` clause. You want to display the subcategories of the category in the first drop-down list, which means you want to match on the `ProductCategory` column. Specifically, you want to select the rows where the `ProductCategoryID` column is the same as whatever the user selected in `ddlCategory`. The Wizard can do that for you. Select ProductCategoryID for the Column, = for the Operator, and Control for the Source. The Control ID field on the right becomes visible; select `ddlCategory`. You don't need a default value because ddlCategory will always have some value. When you have all that set, your WHERE clause dialog will look like Figure D-5.
3. Click Add to add the WHERE clause; then, click OK to return to the Wizard. On the Wizard, click Next, and then Finish. Remember to go to Source view and add the `[Production]` schema.
4. You've got the data, so now you need to tell the drop-down list how to use it. Click `ddlSubcategory` and select Choose Data Source. When the dialog appears, select `sdsSubcategorySource` for the data source, then `Name` for the field to display, and finally `ProductSubcategoryID` for the field to use for the value. Click OK.
5. Back in Design view, click `ddlCategory`, and set its `AutoPostBack` property to `true`. You want `ddlSubcategory` to change as soon as the user makes a selection in `ddlCategory`, so you want it to post back right away.
6. Save everything and test it. When you make a selection in the Category drop-down list, the Subcategory drop-down list should change automatically. If it doesn't, go back and check the `WHERE` statement for `ddlSubcategory`.

Two drop-down lists are now set; time for the third.

1. Add another `SqlDataSource` (`sdsColorSource`) and another `DropDownList` (`ddlColor`). Add a `Label`, too. It would be nice if AdventureWorks had a `Color` table with colors and IDs for each, but it doesn't. The colors are stored in the `Product` table, so you need to retrieve them from there. That's not hard, but it requires a little SQL trick to make it look good. Configure `sdsColorSource` to

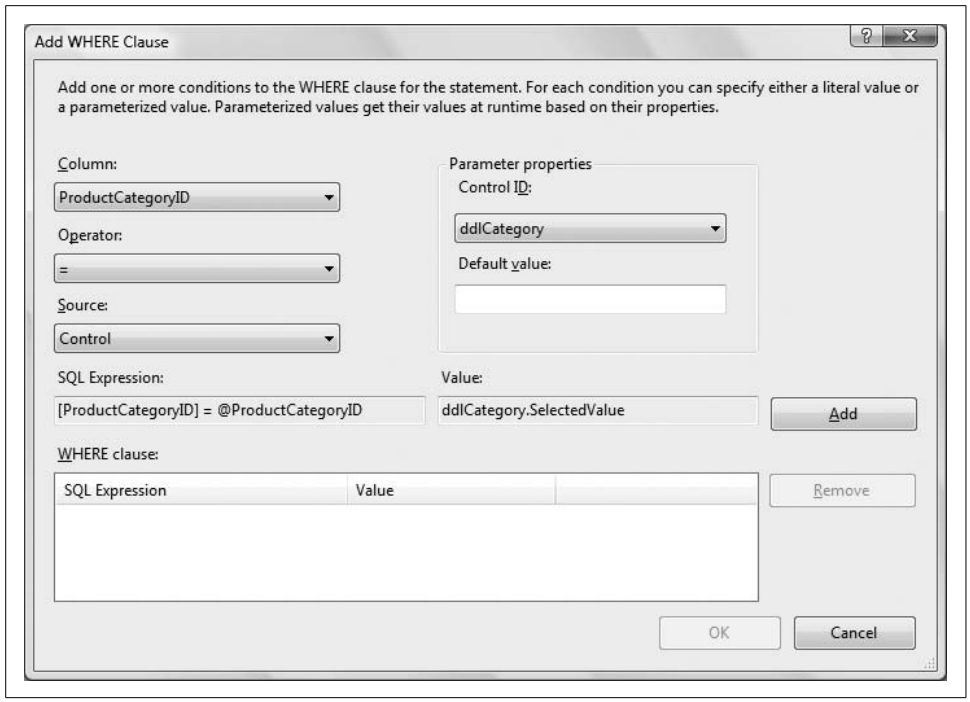


Figure D-5. *Setting the WHERE clause for ddlSubcategory.*

retrieve just the `Color` column from the Product table—no need for WHERE clauses this time. Remember to add the `[Production]` schema in Source view.

2. Click `ddlColor` and select Choose Data Source. In the dialog, select `sdsColorSource` for the data source, and select Color for the field and the value to display. Click OK.
3. If you were to run your application right now, you'd see that `ddlColor` would have a long list of colors—many repeated, and many blank. This actually works fine, but it doesn't look very good. What you want is to only retrieve colors that aren't already in the list. The easiest way to do that is to use an extra bit of SQL code called `DISTINCT`. `DISTINCT` does exactly what it sounds like—it makes sure that each data item in the list is unique, by discarding the repeats. To make this work, switch to source view and find this line, which contains the Select statement for `sdsColorSource`:

   ```
   SelectCommand="SELECT [Color] FROM [Production].[Product]"></asp:SqlDataSource>
   ```

   Just add `DISTINCT`, like this:

   ```
   SelectCommand="SELECT DISTINCT [Color] FROM _
   [Production].[Product]"></asp:SqlDataSource>
   ```

4. Now test your application. You should see a much shorter list in `ddlColor`, and each item is unique.

Now you have all the tools users need to make a selection, so all you need to do is give them a way to see the results.

1. Add one more `SqlDataSource` (call it `sdsProducts`), and a `GridView` (`gvProducts`).
2. Configure the data source to retrieve the `ProductID`, `Name`, `ProductNumber`, and `Color` from the `Product` table.
3. Click the WHERE button. This time, you want two `WHERE` clauses: one for the subcategory, and one for the color. (The category doesn't matter; it's only there to populate the subcategory list.) For the first `WHERE`, set the Column to ProductSubcategoryID, the Operation to =, and the Source to Control. On the right, set the Control ID to ddlSubcategory. Click Add.
4. For the second `WHERE` clause, set the Column to Color, the Operation to =, and the Source to Control. On the right, set the Control ID to `ddlColor`. The dialog should look like Figure D-6.

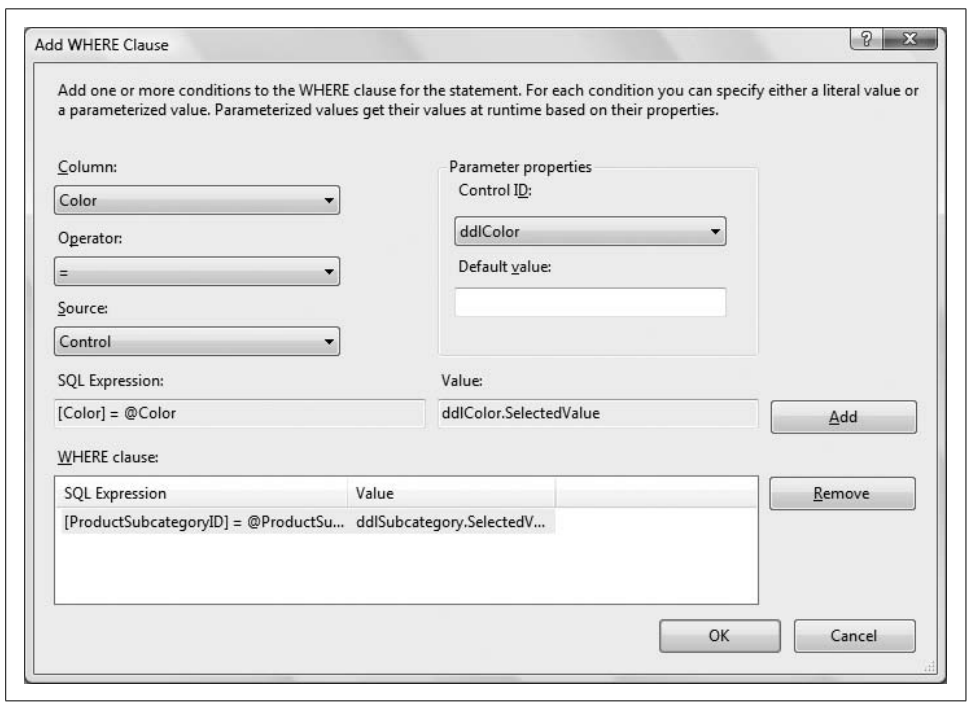


Figure D-6. *Adding the WHERE clauses for the GridView. Notice that the subcategory WHERE clause is already added.*

5. Click Add to add the second WHERE, and OK to return to the Wizard. Click Next and Finish in the Wizard; then, switch to Source view to add the `[Production]` schema.
6. Back in Design view, click `gvProducts`, select `sdsProducts` as the data source, and enable sorting and paging. Apply a nice format if you like.

You don't want to set either ddlSubcategory or ddlColor to auto-postback because you want users to be able to make both choices before the GridView does anything. The drop-down lists have AutoPostBack set to false by default, so that's no problem. You need something to trigger the postback, though, so add a Button and change its text to "Submit." Buttons are set to post back whenever they're clicked, and that's all you want it to do, so you don't even need an event handler for this button. Easy!

Run your application and try it out. Try selecting black gloves or yellow jerseys to see the full effect. Note that if there are no items in the color you've chosen, the GridView will not appear.

Well done! You've taken a long and confusing product list, and with just a handful of drop-down lists, you've made it much friendlier and easier for readers to navigate. With the selection tools you learned about in this chapter, you can imagine that you could enable the user to select items from the GridView to see product details, or add it to a shopping cart, but that's a subject for later on.

Example D-20 shows the markup file for Exercise 4-4.

Example D-20. *The markup file for Exercise 4-4*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 4-4</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:SqlDataSource ID="sdsCategorySource" runat="server"
            ConnectionString=
                "<%$ ConnectionStrings:AdventureWorksConnectionString %>"
            SelectCommand="SELECT [ProductCategoryID], [Name]
                           FROM [Production].[ProductCategory]">
        </asp:SqlDataSource>
        <asp:SqlDataSource ID="sdsSubcategorySource" runat="server"
            ConnectionString=
                "<%$ ConnectionStrings:AdventureWorksConnectionString %>"
            SelectCommand="SELECT [ProductSubcategoryID], [Name]
                           FROM [Production].[ProductSubcategory]
                           WHERE ([ProductCategoryID] = @ProductCategoryID)">
            <SelectParameters>
                <asp:ControlParameter ControlID="ddlCategory"
                    Name="ProductCategoryID"
                    PropertyName="SelectedValue" Type="Int32" />
            </SelectParameters>
        </asp:SqlDataSource>
```

Example D-20. *The markup file for Exercise 4-4 (continued)*

```
    <asp:SqlDataSource ID="sdsColorSource" runat="server"
        ConnectionString=
            "<%$ ConnectionStrings:AdventureWorksConnectionString %>"
        SelectCommand="SELECT DISTINCT [Color]
                       FROM [Production].[Product]"></asp:SqlDataSource>
    <asp:SqlDataSource ID="sdsProducts" runat="server"
        ConnectionString=
            "<%$ ConnectionStrings:AdventureWorksConnectionString %>"

        SelectCommand="SELECT [ProductID], [Name], [ProductNumber], [Color]
                       FROM [Production].[Product]
                       WHERE (([ProductSubcategoryID] = @ProductSubcategoryID)
                       AND ([Color] = @Color))">
        <SelectParameters>
            <asp:ControlParameter ControlID="ddlSubcategory"
                Name="ProductSubcategoryID"
                PropertyName="SelectedValue" Type="Int32" />
            <asp:ControlParameter ControlID="ddlColor" Name="Color"
                PropertyName="SelectedValue" Type="String" />
        </SelectParameters>
    </asp:SqlDataSource>
    <asp:Label ID="lblCategory" runat="server" Font-Bold="True"
                  Text="Category: "></asp:Label>
    <asp:DropDownList ID="ddlCategory" runat="server"
        DataSourceID="sdsCategorySource" DataTextField="Name"
        DataValueField="ProductCategoryID" AutoPostBack="True">
    </asp:DropDownList>

    <asp:Label ID="lblSubcategory" runat="server" Font-Bold="True"
        Text="Subcategory: "></asp:Label>
    <asp:DropDownList ID="ddlSubcategory" runat="server"
        DataSourceID="sdsSubcategorySource" DataTextField="Name"
        DataValueField="ProductSubcategoryID">
    </asp:DropDownList>

    <asp:Label ID="lblColor" runat="server" Font-Bold="True"
         Text="Color:"></asp:Label>
    <asp:DropDownList ID="ddlColor" runat="server"
        DataSourceID="sdsColorSource"
        DataTextField="Color" DataValueField="Color">
    </asp:DropDownList>

    <asp:Button ID="btnSubmit" runat="server" Text="Submit" />

    <br />

    <asp:GridView ID="gvProducts" runat="server" AllowPaging="True"
        AllowSorting="True" CellPadding="4" DataSourceID="sdsProducts"
        ForeColor="#333333" GridLines="None">
        <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
        <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
        <PagerStyle BackColor="#284775" ForeColor="White" HorizontalAlign="Center" />
```

Example D-20. *The markup file for Exercise 4-4 (continued)*

```
            <SelectedRowStyle BackColor="#E2DED6" Font-Bold="True" ForeColor="#333333" />
            <HeaderStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
            <EditRowStyle BackColor="#999999" />
            <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
        </asp:GridView>

    </div>
    </form>
</body>
</html>
```

Chapter 5: Validation

Answers to Quiz Questions

1. You need validation because users make mistakes. The wrong input could result in misplaced orders, inaccurate records, and can even corrupt your database. Validation gets the user to fix those errors before they get anywhere near your data.
2. Set the button's `CausesValidation` property to `false`.
3. The `RequiredFieldValidator`—in the case of a radio button list, you've already defined the choices for the user, so you don't need to validate the form or type of input; you just need to make sure that they chose something.
4. When the `Display` property is set to `Static`, the validator takes up a fixed amount of room, even if it's not displaying a message. When it's set to `Dynamic`, the control is only rendered when there's a validation error, which can cause other controls to move around.
5. Use a `RequiredFieldValidator`, and set the `InitialValue` property to "Choose a payment method."
6. The `ValidationSummary` control enables you to place all the validation error messages in one spot on the page, instead of next to each control.
7. Use a `CompareValidator`. You can compare the quantity the user ordered with the amount of inventory in your database, and make sure that the amount the user wants is equal to or less than the amount you have.
8. Use a `NoSnoreValidation` control to make sure none of the guests snore too loudly. Just kidding. Use a `RangeValidator`. Set the `MinimumValue` to `2`, and the `MaximumValue` to `5`.
9. Use a `RegularExpressionValidator`. The Regular Expression Editor has an option that provides you with a regular expression to validate the form of an email address.

10. You'd need to use a `CustomValidator`. The `RangeValidator` can only check values in a single range. With a `CustomValidator`, though, you could write code to check that the age the user entered either falls between 6 and 12, or is greater than 65.

Answers to Exercises

Exercise 5-1. This one is fairly simple to start out with. Just make a table with five labels, five textboxes, and five Required Field Validators. We specified that you didn't need to worry about the format of the data, so a `RequiredFieldvalidator` is all you need right now. Just make sure you have the validators targeting the correct controls, add some appropriate error messages, and you're done. Example D-21 shows the markup file for this exercise.

Example D-21. *The markup file for Exercise 5-1*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 5-1</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <h1>
        Phone Survey Participation Form</h1>
            <table>
                <tr>
                    <td style="text-align: right">
                        <asp:Label ID="lblName" runat="server" Text="Name:">
                        </asp:Label>
                    </td>
                    <td style="width: 100">
                        <asp:TextBox ID="txtName" runat="server">
                        </asp:TextBox>
                    </td>
                    <td style="width: 100">
                        <asp:RequiredFieldValidator ID="rfvName" runat="server"
                            ControlToValidate="txtName"
                            ErrorMessage="RequiredFieldValidator">
                                Please enter your name
                        </asp:RequiredFieldValidator>
                    </td>
                </tr>
                <tr>
                    <td style="text-align: right">
```

Example D-21. *The markup file for Exercise 5-1 (continued)*

```
            <asp:Label ID="lblAddress" runat="server"
                 Text="Street address:">
            </asp:Label>
        </td>
        <td>
            <asp:TextBox ID="txtAddress" runat="server"></asp:TextBox>
        </td>
        <td>
            <asp:RequiredFieldValidator ID="rfvAddress" runat="server"
                ControlToValidate="txtAddress"
                ErrorMessage="RequiredFieldValidator">
                    Please enter your address
            </asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <td style="text-align: right">
            <asp:Label ID="lblCity" runat="server" Text="City:">
            </asp:Label>
        </td>
        <td>
            <asp:TextBox ID="txtCity" runat="server">
            </asp:TextBox>
        </td>
        <td>
            <asp:RequiredFieldValidator ID="rfvCity" runat="server"
                ControlToValidate="txtCity"
                ErrorMessage="RequiredFieldValidator">
                    Please enter your city
            </asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <td style="text-align: right">
            <asp:Label ID="lblState" runat="server" Text="State:">
            </asp:Label>
        </td>
        <td>
            <asp:TextBox ID="txtState" runat="server">
            </asp:TextBox>
        </td>
        <td>
            <asp:RequiredFieldValidator ID="rfvState" runat="server"
                ControlToValidate="txtState"
                ErrorMessage="RequiredFieldValidator">
                    Please enter your state
            </asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <td style="text-align: right">
            <asp:Label ID="lblZip" runat="server" Text="ZIP code">
            </asp:Label>
        </td>
```

Example D-21. *The markup file for Exercise 5-1 (continued)*

```
                    <td>
                        <asp:TextBox ID="txtZip" runat="server">
                        </asp:TextBox>
                    </td>
                    <td>
                        <asp:RequiredFieldValidator ID="rfvZip" runat="server"
                            ControlToValidate="txtZip"
                            ErrorMessage="RequiredFieldValidator">
                            Please enter your ZIP code
                        </asp:RequiredFieldValidator>
                    </td>
                </tr>
                <tr>
                    <td>
                         </td>
                    <td style="text-align: center">
                        <asp:Button ID="btnSubmit" runat="server"
                             Text="Submit" />
                    </td>
                    <td>
                         </td>
                </tr>
            </table>

    </div>
    </form>
</body>
</html>
```

Exercise 5-2. This exercise also isn't too difficult. You saw how to add a `ValidationSummary` control earlier in the chapter. All you need to do is change the text of the other validator controls to `ErrorMessage` properties, and the messages will display automatically in the summary control.

To check the user's age, you need two controls: a `RequiredFieldValidator`, to make sure the user can't leave the field blank, and a `CompareValidator` to make sure that the user is over 18 (or that they say they are, anyway). The `RequiredFieldValidator` is the same as the ones in Exercise 5-1. For the `CompareValidator`, be sure to set the `ControlToValidate` property to `txtAge` (or whatever you called the age `TextBox`), set the `Type` property to `Integer`, set the `Operator` property to `GreaterThanEqual`, and set the `ValueToCompare` property to 18.

You want to make sure that the user enters a date when they'd like you to call, and that the date is sometime in July. Sounds like a job for the `RangeValidator`. You need to make sure the user enters a date, so add a `RequiredFieldValidator` that validates `txtCallDate` first. Then, add a `RangeValidator` that also validates `txtCallDate`. Make sure you set the `Type` to `Date`, or you'll get unexpected results. Set the `MaximumValue` to 07/31/2009 and the `MinimumValue` to 07/01/2009. Add appropriate text and error messages, and you're done. Try it out, and you'll see that the user can only enter dates in July. Example D-22 shows the markup file for this exercise.

Example D-22. *The markup file for Exercise 5-2*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 5-2</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <h1>
        Phone Survey Participation Form</h1>
            <table>
                <tr>
                    <td style="text-align: right">
                        <asp:Label ID="lblName" runat="server" Text="Name:">
                        </asp:Label>
                    </td>
                    <td>
                        <asp:TextBox ID="txtName" runat="server">
                        </asp:TextBox>
                    </td>
                    <td>
                        <asp:RequiredFieldValidator ID="rfvName" runat="server"
                            ControlToValidate="txtName"
                            ErrorMessage="Please enter your name">
                            *</asp:RequiredFieldValidator>
                    </td>
                </tr>
                <tr>
                    <td style="text-align: right">
                        <asp:Label ID="lblAddress" runat="server"
                             Text="Street address:">
                        </asp:Label>
                    </td>
                    <td>
                        <asp:TextBox ID="txtAddress" runat="server">
                        </asp:TextBox>
                    </td>
                    <td>
                        <asp:RequiredFieldValidator ID="rfvAddress" runat="server"
                            ControlToValidate="txtAddress"
                            ErrorMessage="Please enter your address">
                            *</asp:RequiredFieldValidator>
                    </td>
                </tr>
                <tr>
                    <td style="text-align: right">
```

Example D-22. *The markup file for Exercise 5-2 (continued)*

```
            <asp:Label ID="lblCity" runat="server" Text="City:">
            </asp:Label>
        </td>
        <td>
            <asp:TextBox ID="txtCity" runat="server">
            </asp:TextBox>
        </td>
        <td>
            <asp:RequiredFieldValidator ID="rfvCity" runat="server"
                ControlToValidate="txtCity"
                ErrorMessage="Please enter your city">
                *</asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <td style="text-align: right">
            <asp:Label ID="lblState" runat="server" Text="State:">
            </asp:Label>
        </td>
        <td>
            <asp:TextBox ID="txtState" runat="server">
            </asp:TextBox>
        </td>
        <td>
            <asp:RequiredFieldValidator ID="rfvState" runat="server"
                ControlToValidate="txtState"
                ErrorMessage="Please enter your state">
                *</asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <td style="text-align: right">
            <asp:Label ID="lblZip" runat="server" Text="ZIP code">
            </asp:Label>
        </td>
        <td>
            <asp:TextBox ID="txtZip" runat="server">
            </asp:TextBox>
        </td>
        <td>
            <asp:RequiredFieldValidator ID="rfvZip" runat="server"
                ControlToValidate="txtZip"
                ErrorMessage="Please enter your ZIP code">
                *</asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <td style="text-align: right">
            <asp:Label ID="lblAge" runat="server" Text="Age:">
            </asp:Label>
        </td>
        <td>
```

Example D-22. *The markup file for Exercise 5-2 (continued)*

```
            <asp:TextBox ID="txtAge" runat="server">
            </asp:TextBox>
        </td>
        <td>
            <asp:RequiredFieldValidator ID="rfvAge" runat="server"
                ControlToValidate="txtAge"
                ErrorMessage="Please enter your age">
                *</asp:RequiredFieldValidator>
            <asp:CompareValidator ID="cvAge" runat="server"
                ControlToValidate="txtAge"
                ErrorMessage="You must be over 18 to participate"
                Operator="GreaterThanEqual"
                Type="Integer"
                ValueToCompare="18">
                *</asp:CompareValidator>
        </td>
    </tr>
    <tr>
        <td style="text-align: right">
            <asp:Label ID="lblCallDate" runat="server" Width="150"
                Text="Enter a date for us to call you, in July, 2009
                     (format mm/dd/yyyy):">
           </asp:Label>
        </td>
        <td>
            <asp:TextBox ID="txtCallDate" runat="server">
            </asp:TextBox>
        </td>
        <td>
            <asp:RequiredFieldValidator ID="rfvCallDate" runat="server"
                ControlToValidate="txtCallDate"
                ErrorMessage="Please enter a date for your call">
                *</asp:RequiredFieldValidator>
            <asp:RangeValidator ID="rvCallDate" runat="server"
                ControlToValidate="txtCallDate"
               ErrorMessage="The date must be between 7/1/2009 and 7/31/2009"
                MaximumValue="7/31/2009"
                MinimumValue="7/1/2009"
                Type="Date">
                *</asp:RangeValidator>
        </td>
    </tr>
    <tr>
        <td colspan="3">
            <asp:ValidationSummary ID="vsSummary" runat="server"
                HeaderText="The following error(s) were found:"
                style="text-align: center" />
        </td>
    </tr>
    <tr>
        <td colspan="3" style="text-align: center">
            <asp:Button ID="btnSubmit" runat="server"
```

Example D-22. *The markup file for Exercise 5-2 (continued)*

```
                        Text="Submit" />
                </td>
            </tr>
        </table>
    </div>
    </form>
</body>
</html>
```

Exercise 5-3. This time around, you need three validators on a single `TextBox`. The `RequiredFieldValidator` and `RangeValidator` work the same as they do in Exercise 5-2—you can even copy-and-paste the appropriate markup in Source view to make the new row. However, be sure to change the `ControlToValidate` properties of both validators to `txtFollowup` (or whatever you call the follow-up text box).

Now, you need to add a third validator—this time, a `CompareValidator`. You're not comparing to a constant value, though, like you were with the age text box. This time, you want to make sure that the date in `txtFollowup` is later than the date in `txtCallDate`, which means you want the value in `txtFollowup` to be greater. Add your `CompareValidator`, set its `ControlToValidate` property to `txtFollowup`, and set the `ControlToCompare` property to `txtCallDate`. Be sure not to get the two properties backward, or you'll get unexpected results. Now, set the `Operator` property to `GreaterThan`, and make sure to set the `Type` to `Date`. Add the appropriate text and error messages, and try it out. You'll see that the follow-up date still has to be in July, but also that it must be later than the date of the original call. (If you want to be extra helpful, you can change the `RangeValidator` for `txtCallDate` so that the latest date is July 30, which leaves time for a follow-up on July 31.) Example D-23 shows the markup file for Exercise 5-3.

Example D-23. *The markup file for Exercise 5-3*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 5-3</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    <h1>
        Phone Survey Participation Form</h1>
            <table>
                <tr>
```

Example D-23. *The markup file for Exercise 5-3 (continued)*

```
            <td style="text-align: right">
                <asp:Label ID="lblName" runat="server" Text="Name:">
                </asp:Label>
            </td>
            <td>
                <asp:TextBox ID="txtName" runat="server">
                </asp:TextBox>
            </td>
            <td>
                <asp:RequiredFieldValidator ID="rfvName" runat="server"
                    ControlToValidate="txtName"
                    ErrorMessage="Please enter your name">
                    *</asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <td style="text-align: right">
                <asp:Label ID="lblAddress" runat="server"
                 Text="Street address:">
                </asp:Label>
            </td>
            <td>
                <asp:TextBox ID="txtAddress" runat="server">
                </asp:TextBox>
            </td>
            <td>
                <asp:RequiredFieldValidator ID="rfvAddress" runat="server"
                    ControlToValidate="txtAddress"
                    ErrorMessage="Please enter your address">
                    *</asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <td style="text-align: right">
                <asp:Label ID="lblCity" runat="server" Text="City:">
                </asp:Label>
            </td>
            <td>
                <asp:TextBox ID="txtCity" runat="server">
                </asp:TextBox>
            </td>
            <td>
                <asp:RequiredFieldValidator ID="rfvCity" runat="server"
                    ControlToValidate="txtCity"
                    ErrorMessage="Please enter your city">
                    *</asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <td style="text-align: right">
                <asp:Label ID="lblState" runat="server" Text="State:">
                </asp:Label>
            </td>
```

Example D-23. *The markup file for Exercise 5-3 (continued)*

```
            <td>
                <asp:TextBox ID="txtState" runat="server">
                </asp:TextBox>
            </td>
            <td>
                <asp:RequiredFieldValidator ID="rfvState" runat="server"
                    ControlToValidate="txtState"
                    ErrorMessage="Please enter your state">
                    *</asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <td style="text-align: right">
                <asp:Label ID="lblZip" runat="server" Text="ZIP code">
                </asp:Label>
            </td>
            <td>
                <asp:TextBox ID="txtZip" runat="server">
                </asp:TextBox>
            </td>
            <td>
                <asp:RequiredFieldValidator ID="rfvZip" runat="server"
                    ControlToValidate="txtZip"
                    ErrorMessage="Please enter your ZIP code">
                    *</asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <td style="text-align: right">
                <asp:Label ID="lblAge" runat="server" Text="Age:">
                </asp:Label>
            </td>
            <td>
                <asp:TextBox ID="txtAge" runat="server">
                </asp:TextBox>
            </td>
            <td>
                <asp:RequiredFieldValidator ID="rfvAge" runat="server"
                    ControlToValidate="txtAge"
                    ErrorMessage="Please enter your age">
                    *</asp:RequiredFieldValidator>
                <asp:CompareValidator ID="cvAge" runat="server"
                    ControlToValidate="txtAge"
                    ErrorMessage="You must be over 18 to participate"
                    Operator="GreaterThanEqual"
                    Type="Integer"
                    ValueToCompare="18">
                    *</asp:CompareValidator>
            </td>
        </tr>
        <tr>
            <td style="text-align: right">
```

Example D-23. *The markup file for Exercise 5-3 (continued)*

```
                <asp:Label ID="lblCallDate" runat="server" Width="150"
                    Text="Enter a date for us to call you, in July, 2009
                          (format mm/dd/yyyy):"></asp:Label>
            </td>
            <td>
                <asp:TextBox ID="txtCallDate" runat="server">
                </asp:TextBox>
            </td>
            <td>
                <asp:RequiredFieldValidator ID="rfvCallDate" runat="server"
                    ControlToValidate="txtCallDate"
                    ErrorMessage="Please enter a date for your call">
                    *</asp:RequiredFieldValidator>
                <asp:RangeValidator ID="rvCallDate" runat="server"
                    ControlToValidate="txtCallDate"
                   ErrorMessage="The date must be between 7/1/2009 and 7/30/2009"
                    MaximumValue="7/30/2009" MinimumValue="7/1/2009"
                    Type="Date">
                    *</asp:RangeValidator>
            </td>
        </tr>
        <tr>
            <td style="text-align: right">
                <asp:Label ID="lblFollowup" runat="server" Width="150"
                    Text="Enter a date for us to make a follow-up call,
                          in July, 2009 (format mm/dd/yyyy):">
                </asp:Label>
            </td>
            <td>
                <asp:TextBox ID="txtFollowup" runat="server">
                </asp:TextBox>
            </td>
            <td>
                <asp:RequiredFieldValidator ID="rfvFollowup" runat="server"
                    ControlToValidate="txtFollowup"
                    ErrorMessage="Please enter a date for your follow-up call">
                    *</asp:RequiredFieldValidator>
                <asp:RangeValidator ID="rvFollowup" runat="server"
                    ControlToValidate="txtFollowup"
                   ErrorMessage="The date must be between 7/1/2009 and 7/31/2009"
                    MaximumValue="7/31/2009" MinimumValue="7/1/2009"
                    Type="Date">
                    *</asp:RangeValidator>
                <asp:CompareValidator ID="cvFollowup" runat="server"
                    ErrorMessage="Please select a date after your first call"
                   ControlToCompare="txtCallDate" ControlToValidate="txtFollowup"
                    Operator="GreaterThan"
                    Type="Date">
                    *</asp:CompareValidator>
            </td>
        </tr>
```

Example D-23. *The markup file for Exercise 5-3 (continued)*

```
                <tr>
                    <td colspan="3">
                        <asp:ValidationSummary ID="vsSummary" runat="server"
                            HeaderText="The following error(s) were found:"
                            style="text-align: center" />
                    </td>
                </tr>
                <tr>
                    <td colspan="3" style="text-align: center">
                        <asp:Button ID="btnSubmit" runat="server"
                         Text="Submit" />
                    </td>
                </tr>
            </table>

    </div>
    </form>
</body>
</html>
```

Exercise 5-4. Adding another couple of rows to the table is old hat for you by now, and so is adding the `RequiredFieldValidator` controls for each of the new text boxes. To check the format for the phone number and email address, though, you'll need to use a `RegularExpressionValidator` for each. This isn't too tricky: for the phone number, drag a `RegularExpressionValidator` next to the `RequiredFieldValidator`. Set its `ControlToValidate` to `txtPhone` (or whatever you called the text box), click the `ValidationExpression` property, and then click the ellipsis button to open the Regular Expression Editor. Select "U.S. phone number" from the list (or whatever country you'd like to validate for), and click OK. Add text and error message properties, and it's good to go. The `RegularExpressionValidator` for the email field works the same way, except that the `ControlToValidate` is `txtEmail` (or whatever you called that text box), and you select "Internet e-mail address" in the Regular Expression Editor. Try it out, and you'll see that the page works. You can try a few different variations on phone numbers, and the most common ones will be accepted. Example D-24 shows the markup for Exercise 5-4.

Example D-24. *The markup file for Exercise 5-4*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Exercise 5-4</title>
</head>
<body>
    <form id="form1" runat="server">
```

Example D-24. *The markup file for Exercise 5-4 (continued)*

```
    <div>

    <h1>
        Phone Survey Participation Form</h1>
            <table>
                <tr>
                    <td style="text-align: right">
                        <asp:Label ID="lblName" runat="server" Text="Name:">
                        </asp:Label>
                    </td>
                    <td>
                        <asp:TextBox ID="txtName" runat="server">
                        </asp:TextBox>
                    </td>
                    <td>
                        <asp:RequiredFieldValidator ID="rfvName" runat="server"
                            ControlToValidate="txtName"
                            ErrorMessage="Please enter your name">
                            *</asp:RequiredFieldValidator>
                    </td>
                </tr>
                <tr>
                    <td style="text-align: right">
                        <asp:Label ID="lblAddress" runat="server"
                         Text="Street address:">
                        </asp:Label>
                    </td>
                    <td>
                        <asp:TextBox ID="txtAddress" runat="server">
                        </asp:TextBox>
                    </td>
                    <td>
                        <asp:RequiredFieldValidator ID="rfvAddress" runat="server"
                            ControlToValidate="txtAddress"
                            ErrorMessage="Please enter your address">
                            *</asp:RequiredFieldValidator>
                    </td>
                </tr>
                <tr>
                    <td style="text-align: right">
                        <asp:Label ID="lblCity" runat="server" Text="City:">
                        </asp:Label>
                    </td>
                    <td>
                        <asp:TextBox ID="txtCity" runat="server">
                        </asp:TextBox>
                    </td>
                    <td>
                        <asp:RequiredFieldValidator ID="rfvCity" runat="server"
                            ControlToValidate="txtCity"
                            ErrorMessage="Please enter your city">
                            *</asp:RequiredFieldValidator>
                    </td>
                </tr>
```

Example D-24. *The markup file for Exercise 5-4 (continued)*

```
<tr>
    <td style="text-align: right">
        <asp:Label ID="lblState" runat="server" Text="State:">
        </asp:Label>
    </td>
    <td>
        <asp:TextBox ID="txtState" runat="server">
        </asp:TextBox>
    </td>
    <td>
        <asp:RequiredFieldValidator ID="rfvState" runat="server"
            ControlToValidate="txtState"
            ErrorMessage="Please enter your state">
            *</asp:RequiredFieldValidator>
    </td>
</tr>
<tr>
    <td style="text-align: right">
        <asp:Label ID="lblZip" runat="server" Text="ZIP code">
        </asp:Label>
    </td>
    <td>
        <asp:TextBox ID="txtZip" runat="server">
        </asp:TextBox>
    </td>
    <td>
        <asp:RequiredFieldValidator ID="rfvZip" runat="server"
            ControlToValidate="txtZip"
            ErrorMessage="Please enter your ZIP code">
            *</asp:RequiredFieldValidator>
    </td>
</tr>
<tr>
    <td style="text-align: right">
        <asp:Label ID="lblAge" runat="server" Text="Age:">
        </asp:Label>
    </td>
    <td>
        <asp:TextBox ID="txtAge" runat="server">
        </asp:TextBox>
    </td>
    <td>
        <asp:RequiredFieldValidator ID="rfvAge" runat="server"
            ControlToValidate="txtAge"
            ErrorMessage="Please enter your age">
            *</asp:RequiredFieldValidator>
        <asp:CompareValidator ID="cvAge" runat="server"
            ControlToValidate="txtAge"
            ErrorMessage="You must be over 18 to participate"
            Operator="GreaterThanEqual"
            Type="Integer"
```

Example D-24. *The markup file for Exercise 5-4 (continued)*

```
                ValueToCompare="18">
                *</asp:CompareValidator>
        </td>
    </tr>
    <tr>
        <td style="text-align: right">
            <asp:Label ID="lblPhone" runat="server" Text="Phone:">
            </asp:Label>
        </td>
        <td>
            <asp:TextBox ID="txtPhone" runat="server">
            </asp:TextBox>
        </td>
        <td>
            <asp:RequiredFieldValidator ID="rfvPhone" runat="server"
                ControlToValidate="txtPhone"
                ErrorMessage="Please enter your phone number">
                *</asp:RequiredFieldValidator>
            <asp:RegularExpressionValidator ID="revPhone" runat="server"
                ControlToValidate="txtPhone"
                ErrorMessage="Please enter a valid phone number"
                ValidationExpression="((\(\d{3}\) ?)|(\d{3}-))?\d{3}-\d{4}">
                *</asp:RegularExpressionValidator>
        </td>
    </tr>
    <tr>
        <td style="text-align: right">
            <asp:Label ID="lblEmail" runat="server"
                Text="E-mail address:">
            </asp:Label>
        </td>
        <td>
            <asp:TextBox ID="txtEmail" runat="server">
            </asp:TextBox>
        </td>
        <td>
            <asp:RequiredFieldValidator ID="rfvEmail" runat="server"
                ControlToValidate="txtEmail"
                ErrorMessage="Please enter your e-mail address">
                *</asp:RequiredFieldValidator>
            <asp:RegularExpressionValidator ID="revEmail" runat="server"
                ControlToValidate="txtEmail"
                ErrorMessage="Please enter a valide e-mail address"
                ValidationExpression=
                        "\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
            *</asp:RegularExpressionValidator>
        </td>
    </tr>
    <tr>
        <td style="text-align: right">
            <asp:Label ID="lblCallDate" runat="server" Width="150"
```

Example D-24. *The markup file for Exercise 5-4 (continued)*

```
                    Text="Enter a date for us to call you, in July, 2009
                          (format mm/dd/yyyy):">
                </asp:Label>
            </td>
            <td>
                <asp:TextBox ID="txtCallDate" runat="server">
                </asp:TextBox>
            </td>
            <td>
                <asp:RequiredFieldValidator ID="rfvCallDate" runat="server"
                    ControlToValidate="txtCallDate"
                    ErrorMessage="Please enter a date for your call">
                    *</asp:RequiredFieldValidator>
                <asp:RangeValidator ID="rvCallDate" runat="server"
                    ControlToValidate="txtCallDate"
                    ErrorMessage="The date must be between 7/1/2009
                                  and 7/30/2009"
                    MaximumValue="7/30/2009"
                    MinimumValue="7/1/2009"
                    Type="Date">
                    *</asp:RangeValidator>
            </td>
        </tr>
        <tr>
            <td style="text-align: right">
                <asp:Label ID="lblFollowup" runat="server" Width="150"
                    Text="Enter a date for us to make a follow-up call,
                          in July, 2009 (format mm/dd/yyyy):">
                </asp:Label>
            </td>
            <td>
                <asp:TextBox ID="txtFollowup" runat="server">
                </asp:TextBox>
            </td>
            <td>
                <asp:RequiredFieldValidator ID="rfvFollowup" runat="server"
                    ControlToValidate="txtFollowup"
                    ErrorMessage="Please enter a date for your follow-up call">
                    *</asp:RequiredFieldValidator>
                <asp:RangeValidator ID="rvFollowup" runat="server"
                    ControlToValidate="txtFollowup"
                    ErrorMessage="The date must be between 7/1/2009
                                  and 7/31/2009"
                    MaximumValue="7/31/2009"
                    MinimumValue="7/1/2009"
                    Type="Date">
                    *</asp:RangeValidator>
                <asp:CompareValidator ID="cvFollowup" runat="server"
                    ErrorMessage="Please select a date after
                                  your first call"
                    ControlToCompare="txtCallDate"
                    ControlToValidate="txtFollowup"
```

Example D-24. *The markup file for Exercise 5-4 (continued)*

```
                        Operator="GreaterThan"
                        Type="Date">
                        *</asp:CompareValidator>
                </td>
            </tr>
            <tr>
                <td colspan="3">
                    <asp:ValidationSummary ID="vsSummary" runat="server"
                        HeaderText="The following error(s) were found:"
                        style="text-align: center" />
                </td>
            </tr>
            <tr>
                <td colspan="3" style="text-align: center">
                    <asp:Button ID="btnSubmit" runat="server"
                     Text="Submit" />
                </td>
            </tr>
        </table>

    </div>
    </form>
</body>
</html>
```

Chapter 6: Style Sheets, Master Pages, and Navigation

Answers to Quiz Questions

1. The best way to apply styles on your page is to use an external style sheet. You can use inline or document-level styles, but these are error-prone and difficult to maintain.
2. The style for the specific paragraph applies. Style rules are always applied from least to most specific.
3. You can use the `@import` command to apply a style sheet to your page, and it must be placed in the `<head>` element. If you place it anywhere else, the style sheet will be ignored. You can also use a `link` tag, which also must be placed within the `<head>` element. The easiest way to add a `link` tag is by using the Attach Style Sheet link in the Manage Styles window; Visual Studio will add the `link` tag to the appropriate spot for you.
4. A master page acts as a shell, inside of which the content of your individual pages is displayed. This allows you to have a consistent look to all the pages on your site.
5. There's no limit to the number of nested master pages you can associate with a single content page.

6. The child content page needs a reference to the class of the master page, contained in a `@MasterType` directive, immediately following the `@Page` directive for the child page.
7. You can use the `Response.Redirect` method to navigate to another page.
8. You need to add a site map file, which is an XML file. You can use Website → Add New Item to add a new site map with an automatically generated skeleton, but you need to add the content of the XML file yourself.
9. You place a `SiteMapDataSource` control on the page to enable the navigation controls. In the case of a `TreeView` or `Menu` control, you use the Smart Tag to point the control to the `SiteMapDataSource`.
10. You don't have to do anything. The `SiteMapPath` control automatically finds the `SiteMapDataSource` control on the page and uses it.

Answers to Exercises

Exercise 6-1. There are a few different ways to create the pages shown in Exercise 6-1; here's one of them. Create a new web site called Exercise 6-1. Add a new master page called *AjaxTravel.master*, and be sure to put the code in a separate file. Rename the content placeholder something like `cphAjaxMaster`. Add an `<h1>` with the appropriate message, and a footer with the copyright message. Check it out in Design view to make sure it looks the way you want.

Use Website → Add New Item to add a new Web Form called *Home.aspx* to your site, and select *AjaxMaster.master* as its master page. On this page, delete the "head" content placeholder, and add a bit of text welcoming the user, then a `<ul>` with two `<li>` items that each consist of an HTML `<a>` element: one that reads "Sun and surf packages" with an `href` of `Sun.aspx`, and one that reads "Snow and ski packages" with an `href` of `Snow.aspx`.

Add a new master page to the site; call it *SunMaster.master*. As in Example 6-7, remove everything except the `Master` directive.

Next, add a `Content` section with a `ContentPlaceHolderID` attribute of `cphAjaxMaster`. Add some content for the header and footer (a table for layout might be useful), and be sure to include another `ContentPlaceHolder` called `cphSunContent`.

Create another new master page called *SnowMaster.master*, and set it up in the same way as *SunMaster.master*.

Add a new Web Form, and set its master page to be *SunMaster.master*. Create another `<ul>` with links to the Bermuda and Maui pages.

Add a new Web Form, and set its master page to be *SnowMaster.master*. Create another `<ul>` with links to the St. Moritz and Vail pages.

Create content pages for Bermuda and Maui with simple sample text, both of which use *SunMaster.master* as their master page.

Create content pages for St. Moritz and Vail with simple sample text, both of which use *SnowMaster.master* as their master page.

Now you've got a fully functional web site that changes its content depending on where the user wants to go, but maintains brand identity. If this were a real site, of course, you'd use style sheets to jazz up the content and add your company logo and such. Of course, you'd also have real content on the destination pages. You can imagine that you'd probably have a link in the footers to your honeymoon and ski vacation packages that would take the user to the appropriate pages.

From this point on, the sites will be too lengthy to provide full code listings, but you can download the complete sites from this book's web page at *http://www.oreilly.com/catalog/9780596518455*.

Exercise 6-2. Copy Exercise 6-1 to a new web site, Exercise 6-2. Open *AjaxTravel.master*, and add some text underneath the `<h1>`. Type "Welcome," and then drag a `Label` onto the page after the text. Name the label `lblName`, and change its `Text` property to "Guest." Then, add an exclamation point. If you run the site now, you'll see that each page says "Welcome, Guest!"

You need *AjaxTravel.master* to implement a public property to display the label, so open *AjaxTravel.master.vb*, and enter the following code:

```
Public Property MessageLabel( ) As Label
    Get
        Return lblName
    End Get
    Set(ByVal value As Label)
        lblName = value
    End Set
End Property
```

On the *Home.aspx* page, below the `<ul>`, add some text saying "Enter your name:". Then add a Textbox control (`txtName`), and a Submit button. The page needs to know about the `AjaxTravel` page class, so that it can make changes to the master page. Add the following line of code to *Home.aspx*, just after the `Page` directive:

```
<%@ MasterType TypeName="AjaxTravel" %>
```

Now you need to wire up the Submit button so that the text in `txtName` gets copied to the label on the master page. Double-click the Submit button, and when the event handler opens, add the following code:

```
Me.Master.MessageLabel.Text = txtName.Text
```

Try it out. You'll see the control on the home page where you'll be able to enter your name, and it should be transferred to the master page. Unfortunately, if you navigate

to any other page, the greeting goes back to being "Hello, Guest!" To save that information, you'll need to use session state, which you'll see in Chapter 8.

Exercise 6-3. We won't use the user greeting for this web site, so copy Exercise 6-1 to a new web site called Exercise 6-3. The first thing you need to do is add a Site Map, so select Website → Add New Item, then select Site Map, and accept the default name. The skeleton of *web.sitemap* is created for you automatically, but you need to fill in the nodes. The results should look like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="~/Home.aspx" title="Home" description="Home page">
    <siteMapNode url="~/SunHome.aspx" title="Sun" description="Sunny
                                                          destinations">
      <siteMapNode url="~/Bermuda.aspx" title="Bermuda" description="Bermuda" />
      <siteMapNode url="~/Maui.aspx" title="Maui" description="Maui" />
    </siteMapNode>
    <siteMapNode url="~/SnowHome.aspx" title="Snow" description="Snow
                                                          destinations">
      <siteMapNode url="~/StMoritz.aspx" title="St. Moritz" description="St.
                                                                Moritz" />
      <siteMapNode url="~/Vail.aspx" title="Vail" description="Vail, Colorado" />
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

Next, open up *AjaxTravel.master*, and add a `SiteMapDataSource` control to the master page. It doesn't matter where you put it—the data source will find *Web.sitemap* automatically.

Now, add two radio buttons, `rbTree` and `rbMenu`, with Text properties of "Tree View" and "Menu." Be sure to set the `GroupName` property of each control to the same value, such as `grpNavView`. Next, add a `TreeView` control under the radio button list, and set its `Visible` property to `false`. Add a `Menu` control under the `TreeView`, and set its `Visible` property to `false` as well. Set the data source of the `TreeView` and `Menu` controls to point to the `SiteMapDataSource` you added earlier.

Now, you need to set the event handler for the radio buttons. Double-click `rbTree` to be taken to the `CheckChanged` event. You want `TreeView1` to be visible when this button is checked, and `Menu1` to be invisible, so add this code to the event handler:

```
TreeView1.Visible = rbTree.Checked
Menu1.Visible = rbMenu.Checked
```

Create an event handler for the `CheckChanged` event of `rbMenu`, and add the same code.

Now run the site. You should be able to select the `TreeView` or `Menu` navigation control on each page. You may need to rearrange the content of your pages so that the menus will fit. Unfortunately, you can't retain the user's choice from page to page, but again, you'll see how to do that in Chapter 8.

Exercise 6-4. This last exercise is very simple. Simply open *AjaxTravel.master*, and drag a `SiteMapPath` control onto the page. The `SiteMapPath` will automatically find the `SiteMapDataSource` and implement the bread crumbs for you.

Chapter 7: State and Life Cycle

Answers to Quiz Questions

1. A session is the period of time in which a single user interacts with an application, no matter how many individual pages he or she visits.
2. The state of a page refers to the current values of all controls on the page, including any changes made by the user.
3. Add the `Trace="True"` attribute to the `Page` directive to see the page Trace, including the stages of the page life cycle, and the control hierarchy.
4. The postback mode is determined in the Start phase of the life cycle.
5. The `Page_Load` event is the most common event to handle if you want to take actions during the Load phase.
6. ASP.NET manages Control state, View state, Session state, and Application state. You cannot affect the management of the Control state.
7. The `EnableViewState="false"` attribute disables View state for more complex controls. Simpler controls, such as text boxes, retain their state no matter what.
8. Use the state bag to store the value of a counter that increments each time the page is loaded. If you navigate to a separate page, the counter will reset, but not if you click the browser's Refresh button.
9. Save the user's name in session state.
10. Use the syntax `Session("username") = <user's name>`.

Answers to Exercises

Exercise 7-1. The trick to this exercise isn't the code—it's where you put it. You want to evaluate the `IsPostBack` attribute and take action based on its value, so the best place to do that is in the `Page_Load` event. Just a simple bit of code in the `Page_Load` event handler does what you want:

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
    If Not IsPostBack Then
        lblPostBack.Text = "You're seeing this page for the first time!"
    Else
        lblPostBack.Text = "Welcome back to the page."
    End If
End Sub
```

Example D-25 has the complete markup file for Exercise 7-1.

Example D-25. *The markup file for Exercise 7-1*

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Exercise 7-1</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="lblPostBack" runat="server"></asp:Label>
        <br />
        <asp:Button ID="btnPostBack" runat="server" Text="Post Back" />
    </div>
    </form>
</body>
</html>
```

The complete code-behind file is shown in Example D-26.

Example D-26. *The code-behind file for Exercise 7-1*

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles Me.Load
        If Not IsPostBack Then
            lblPostBack.Text = "You're seeing this page for the first time!"
        Else
            lblPostBack.Text = "Welcome back to the page."
        End If

    End Sub
End Class
```

Exercise 7-2. This exercise isn't all that different from Exercise 7-1, except that you need to use the state bag. The button and the label aren't anything special. The key to this exercise is in the code-behind file, specifically, the event handler for the `Page_Load` event.

The first thing you need to do is create an empty string to hold the message that you'll put in the label:

```
    Dim message As String = ""
```

If this is the first time the page has been loaded, you add a message to the string saying that. You use an `If` statement that checks whether the page is a postback, just as in Exercise 7-1. Notice that you use `DateTime.Now` to insert the current time, and then add a line break just to make things look nice:

```
If Not IsPostBack Then
    message += "Page first accessed at " + DateTime.Now + ".<br/>"
End If
```

Then, you assign the string to the `Text` property of the label:

```
lblMessage.Text = message
```

That's all easy enough, but you need that string for the next time the page is posted, so you need to store it in the state bag. You simply create a new item in the dictionary, give it a name, and then assign the string to it:

```
ViewState("message") = message
```

Now, you need to account for when the page is posted back, so you have to go back up and add an `Else` clause to your `If` statement. This time, you want to retrieve the previous message first, so you get it back from the state bag, and use `CType` to convert it to a string. Then, you can add the rest of the message just as you did in the first half of the `If`:

```
Else
   message = CType(ViewState("message"), String) + "Page posted back at " _
             + DateTime.Now + ".<br/>"
```

The statements to assign the message to the label and then store the message back in the state bag happen outside the `If`, so there's no need to repeat them here.

The entire event handler is shown in Example D-27.

Example D-27. *The event handler for the Page_Load event in Exercise 7-3*

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles Me.Load

    Dim message As String = ""
    If Not IsPostBack Then
        message += "Page first accessed at " + DateTime.Now + ".<br/>"
    Else
        message = CType(ViewState("message"), String) + "Page posted back at " _
                  + DateTime.Now + ".<br/>"

    End If
    lblMessage.Text = message
    ViewState("message") = message

End Sub
```

As a bonus, try adding `EnableViewState = "False"` to the `Page` directive, and then run the application again. Instead of appending each new line to the string, every postback will overwrite the existing string.

Exercise 7-3. For this exercise, you need to use the `Session` dictionary instead of the state bag. Start by adding the two new buttons to *Default.aspx*, and give each of them a handler that uses `Response.Redirect` to point to *SecondPage.aspx* and *ThirdPage.aspx*.

You'll need to make some changes to the code-behind file of *Default.aspx*. First, replace the `ViewState` methods with `Session`. In addition, you have to make a change to the first half of the `If` statement:

```
If Not IsPostBack Then
    message += "Page first accessed at " + DateTime.Now + ".<br/>"
```

If you navigate to another page, and then return to the home page, the `message` will be defined again as a blank string. If that happens, you need to retrieve `Session("message")` from the session state, even the first time the page loads, when `Session("message")` will be empty:

```
If Not IsPostBack Then
    message = CType(Session("message"), String) + _
              "Home page first accessed at " + DateTime.Now + ".<br/>"
```

The entirety of the code-behind for *Default.aspx* should now look like Example D-28.

Example D-28. *The code-behind file for Default.aspx in Exercise 7-3*

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load

        Dim message As String = ""
        If Not IsPostBack Then
            message = CType(Session("message"), String) + _
                      "Home page first accessed at " + DateTime.Now + ".<br/>"
        Else
            message = CType(Session("message"), String) + _
                      "Home page posted back at " + DateTime.Now + ".<br/>"
        End If
        lblMessage.Text = message
        Session("message") = message

    End Sub

    Protected Sub btnPage2_Click(ByVal sender As Object, ByVal e As _
    System.EventArgs) Handles btnPage2.Click
        Response.Redirect("SecondPage.aspx")
    End Sub
```

Example D-28. *The code-behind file for Default.aspx in Exercise 7-3 (continued)*

```
    Protected Sub btnPage3_Click(ByVal sender As Object, ByVal e As _
    System.EventArgs) Handles btnPage3.Click
        Response.Redirect("ThirdPage.aspx")
    End Sub
End Class
```

Now, create *SecondPage.aspx* and *ThirdPage.aspx*. These two pages need the same controls as *Default.aspx*, and the same code-behinds, but be sure to change the messages and the `Response.Redirect` targets accordingly.

This time, if you add `EnableViewState = "False"` to the `Page` directive, the application doesn't change because the string is stored in state instead. If you disable session state and try to run the application, you'll get an error.

Chapter 8: Errors, Exceptions, and Bugs, Oh My!

Answers to Quiz Questions

1. Add `Trace="true"` to the `@Page` directive of the page you want to trace.
2. The only difference is that `Trace.Warn` writes to the trace log in red.
3. `Trace.Write` and `Trace.Warn` can both take a category string, a message string, and an exception object.
4. Place the cursor on any line of a code file and click in the left column, or press F9 to set a breakpoint in that line of code. A red dot will appear on the line where you set the breakpoint.
5. Simply hover the mouse over the variable, and a pop-up will appear, showing its value. You can also look at the Locals window.
6. When the application is stopped at a breakpoint, you can use the Immediate window to change the value of a variable.
7. The Locals window shows the variables in the current context and their values.
8. Syntax errors are errors in the code that violate the rules of the language. The IDE can catch most of these for you. Logic errors occur when the syntax of code is correct, but the code does not provide the results that the programmer expected.
9. In the *web.config* file, you need to create a `<customErrors>` section, and set the mode attribute to `On`.
10. Add the `ErrorPage` attribute to the `@Page` directive, where you can specify the error page that will apply only to errors generated by the current page.

Answers to Exercises

Exercise 8-1. After you've downloaded the file, open *Default.aspx*, and add the attribute `Trace="true"` to the `@Page` directive. You've now enabled tracing on this file. Open *Default.aspx.vb*, where you'll find the event handler for the drop-down list. Insert a line of code similar to the following:

```
Trace.Warn("In event handler.")
```

Now run the application. You'll see the trace information immediately, but you won't see your trace message until you select an item from the drop-down list. Once you do, the message will show up in the trace.

Exercise 8-2. In debugging, there are always many different ways to identify a problem. You could try to trace to solve this problem, but it probably won't tell you much. The best thing to do is to set a breakpoint somewhere and have a look at the local variables. The problem is with the text being written to the `Label`, so placing a breakpoint where the text is assigned would be a good idea. Open the *Default.aspx.vb* file, and place a breakpoint on this line:

```
lblProduct.Text = ddlProduct.SelectedItem.Value + "<br />" + description
```

Now run the application. Nothing happens until you make a selection in the drop-down list, which triggers the event handler. Still nothing happens in the application because the breakpoint halted execution before the text was written to the label.

In the Locals window, you should now see the value of several variables. You can see the value of `description` there, but that's not the problem. If you hover over `ddlProduct.SelectedItem.Value`, you'll see that the value is "ox" in this case, which isn't what you want. You might know just from looking at it that `SelectedItem.Value` isn't what you want to write to the label, but if you didn't, there's still more information available. In the Locals window, you'll see that the value of "sender" is {`System.Web.UI.WebControls.DropDownList`}. The problem is with the drop-down list, so that looks promising. If you click the plus sign to expand that item, you'll see another heading at first, but if you click the plus sign next to that new heading, you'll see a list of properties for the `DropDownList`. There's a lot of them, but if you scroll down to "SelectedItem," you'll see that you can expand that, too. Once you do, you'll see that `SelectedItem.Value` is set to "ox," which you don't want, but right above it, `SelectedValue.Text` is set to "Oxford shirt," which is what you want. Therefore, if you change this:

```
lblProduct.Text = ddlProduct.SelectedItem.Value + "<br />" + description
```

to this:

```
lblProduct.Text = ddlProduct.SelectedItem.Text + "<br />" + description
```

you'll resolve the problem.

It's not obvious that you'd find the answer to the problem three levels deep in the Locals window. A large part of debugging is experience, along with knowing where to look. The important part is learning to put the breakpoints at the right spots to give you the best leads you can get.

Exercise 8-3. One of the most frustrating troubleshooting situations is trying to work out why your application is *not* doing something. In this case, you can see that the checkbox is checked, but the panel isn't appearing as expected. Something must be wrong in the event handler. Open *Default.aspx.vb*, and place a breakpoint at the beginning of the event handler:

```
Protected Sub ddlProduct_SelectedIndexChanged(ByVal sender _
As Object, ByVal e As System.EventArgs) Handles ddlProduct.SelectedIndexChanged
```

Run the application. When it stops at the breakpoint, there isn't much to see in the Locals window. If you click the plus sign next to the `Me` item in the Locals window, you'll see a very lengthy list of everything related to the page, including all the controls on it. You need to narrow down the information some. Open the Watch window next to the Locals window (you may need to drag the Watch tab to some other part of the UI to see it). Scroll down in the Locals window until you find `pnlProduct`. Click the plus sign next to it to see the panel's properties. Drag the `Visible` property into the Watch window so you can watch it all by itself.

Now that you know what you're watching, you need the application to continue. Use the Step Into button on the Debugging toolbar, or press F11 to step through the event handler line-by-line. When you reach the end of the event handler, you'll see that the panel's `Visible` property is still `False`—nothing ever changed it. That means that a line is missing from the event handler; specifically, this one:

```
pnlProduct.Visible = cbProduct.Checked
```

Without that line, the panel stays at its initial value of `Visible = False`. Once you insert that line, the application runs properly.

Exercise 8-4. The first thing you need to do is modify the *Web.config* file to indicate that you'll be using a custom error page. Insert the following code between the `<system.web>` tags:

```
<customErrors mode="On" defaultRedirect="Error.htm">
  <error statusCode="404" redirect="Error404.htm"/>
</customErrors>
```

This code specifies a default error page, *Error.htm*, and a page specifically for 404 errors, *Error404.htm*.

The next step is creating error file. Select Website → Add New Item, and select an HTML file, naming it *Error404.htm*. Open the file and add some HTML similar to the following:

```
<html>
<head>
    <title>Bad Link Error</title>
</head>
<body>
      <h1>Error</h1>
      We're sorry, the page you're looking for does not exist. Please notify
      the webmaster.
      Click <a href="Default.aspx">here</a> to return to the product page.
</body>
</html>
```

Return to *Default.aspx*, run the application, and click the link. You'll be taken to your custom error page. You can even use the link to get back to the product page, which probably won't make the user very happy, so it would be a good idea to get the customer assistance page created right away, or to delegate it to a subordinate.

Chapter 9: Security

Answers to Quiz Questions

1. You can create users by hand, using the WAT, or you can allow users to create accounts at runtime with the `CreateUserWizard` control.
2. Forms-based security grants privileges to users based on credentials, such as username and password, which are gathered from the user via a web page. With Windows authentication, user privileges are based on their Windows login, which is more useful for an intranet setup.
3. The security question allows a method for users to identify themselves to the site if the user has forgotten her password. This involves sending an e-mail to the user, and requires the use of an SMTP server.
4. User information is stored in a database named *ASPNETDB.MDF* within the *App_Data* directory of your site.
5. You need to provide a `CreateUserWizard` control so that users can specify their own account information.
6. The `LoginStatus` control has two views: one for anonymous users (users who haven't logged in, or who don't have accounts) and one for logged-in users.
7. The `LoginName` control simply displays the name of the currently logged-in user. This control allows you to personalize the page by addressing the user, even if you don't know who the users are when you design the page.
8. If the user enters an incorrect password, the `Login` control has an automatic message that informs the user of his mistake, and invites the user to try again.
9. You can add users to roles by using the WAT.
10. Use the `User.IsInRole` property to test whether a user is a member of a role before granting access to a page.

Answers to Exercises

Exercise 9-1. Start out by deleting *Default.aspx*, add a new page called *Welcome.aspx*, and set it as the Start page. Add a bit of text if you like, a `LoginStatus` control, and a `LoginView` control. Make any adjustments you like to the text of the `LoginStatus` and `LoginView` controls. To make sure that only logged-in users can access the Guard Schedule page, you can put a `HyperLink` control in the `LoggedInTemplate` of the `LoginView` control. Set the `NavigateUrl` property of the `HyperLink` to *GuardPage.aspx*, or whatever you decide to call the Guard Schedule page.

Next, you need to create the login page. Remember that it must be called *Login.aspx*, and nothing else, or the controls won't be able to find it. Add a bit of text if you like (using master pages would make it easier to extend your brand identity to the entire site, but we won't bother with that here), and add the Login control to the page. Style it how you like, and set the `DestinationPageUrl` property to take users back to *Welcome.aspx*.

Now you have the login mechanism set up, but no users. As we mentioned, you'll need to add the users from the WAT for this exercise, so select Website → ASP.NET Configuration. Click the Security link. You have guards on the day shift and the night shift to create, and you don't want them to access each other's content, so the best way to keep them separate is with roles. To do that, click the "Enable Roles" link on the Security page, then click "Create or Manage Roles." Create one role for the day shift, and one role for the night shift; the administration tool makes that easy. Click the Back button to return to the Security page.

Click the "Select authentication type" link, and switch the authentication from Windows-based to forms-based. Now you're ready to create some users. Click the create user link, and follow the instructions until you've created all four usernames. Make sure that the "active user" box is checked for each one, and also be sure to select the checkbox for the appropriate role for each guard. Go back to the Security page and click the Manage Users link to make sure they're all set up properly. Your page should look something like Figure D-7.

Now that your guards are set up and have their roles, they need some content to look at. Add a new page to your site called *GuardPage.aspx* (or whatever you like; just be sure it's the same name you chose in the `HyperLink` control earlier). You need to have some content in here to indicate placeholders for the guard schedule; it can be a simple bit of text in a `Label`, or a `Panel` control, whatever you like, but you need to be able to control the visibility programmatically. We've chosen to create two tables that show the guard schedules, each within a `Panel` control. The `Visibility` property of each `Panel` is initially set to `false`.

Now you need to create the code to display the appropriate `Panel`, depending on the user's role. Open *GuardPage.aspx.vb*, and create the `Page_Load` handler. You need to use the `User.IsInRole( )` method to determine which role the user is in, and display the appropriate `Panel`. The complete code-behind file is shown in Example D-29.

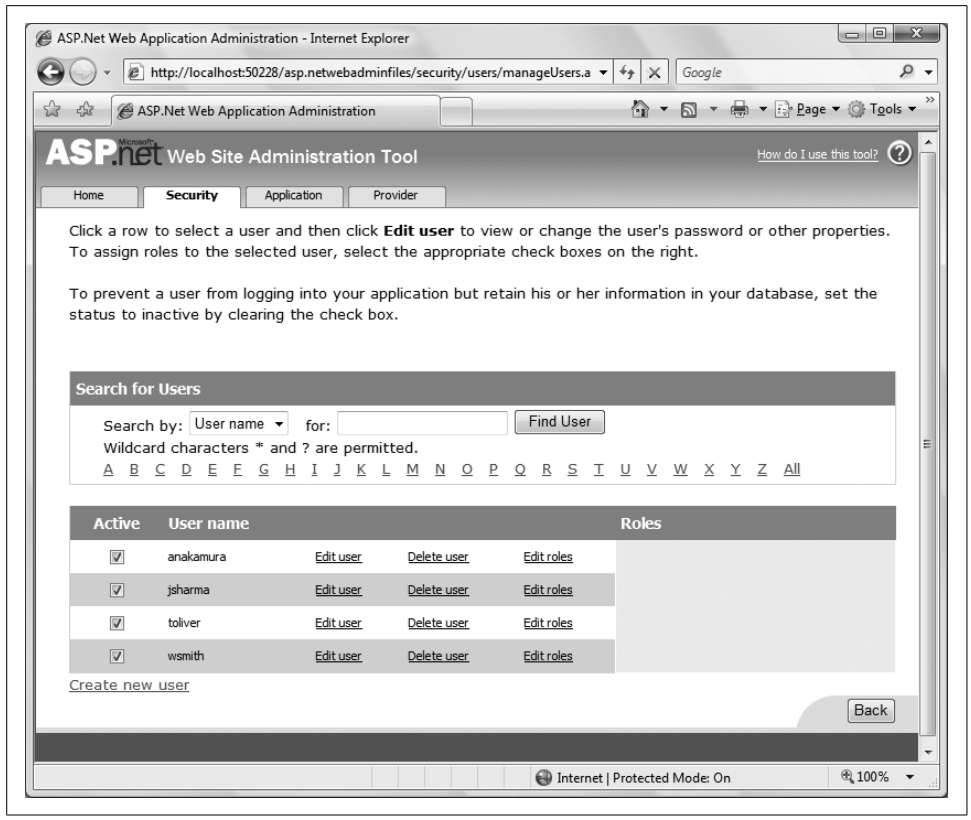

ASP.Net Web Application Administration - Internet Explorer
http://localhost:50228/asp.netwebadminfiles/security/users/manageUsers.a
Google
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Click a row to select a user and then click Edit user to view or change the user's password or other properties.
To assign roles to the selected user, select the appropriate check boxes on the right.
To prevent a user from logging into your application but retain his or her information in your database, set the status to inactive by clearing the check box.
Search for Users
Search by: User name for: Find User
Wildcard characters * and ? are permitted.
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z All
Active
User name
Roles
anakamura Edit user Delete user Edit roles
jsharma Edit user Delete user Edit roles
toliver Edit user Delete user Edit roles
wsmith Edit user Delete user Edit roles
Create new user
Back
Internet | Protected Mode: On
100%

Figure D-7. *The Manage Users page of the Ajax Security site will look like this once you've created user accounts for the four guards.*

Example D-29. *GuardPage.aspx.vb for Exercise 9-1*

```
Partial Class GuardPage
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles Me.Load

        If User.IsInRole("Day shift") Then
            pnlDayShift.Visible = True
            pnl.NightShift.Visible = False
        ElseIf User.IsInRole("Night shift") Then
            pnlNightShift.Visible = True
            pnl.DayShift.Visible = False
        End If

    End Sub
End Class
```

At this point, users who aren't logged in can still access *GuardPage.aspx* by typing in the URL directly, but a user who isn't in either the Day shift or Night shift role (which includes any user who's not logged in) won't be able to see any content on that page.

Exercise 9-2. You can start by copying Exercise 9-1 to a new web site. Now, add a new page, *CustomerInfo.aspx*, to the site. The content of this page doesn't really matter; it's just a placeholder, although you may want to include a link to *Welcome.aspx*. Go back to the *Welcome.aspx* page, and add a link to a new page, *CreateAccount.aspx*. You might want to add this link inside the Anonymous template of the `LoginView` control, so that users only see the link to create a new account if they're not already logged in. While you're here, set *Welcome.aspx* as the Start Page for the site because that setting isn't retained when you copy the web site.

Now that you have a link to the account creation page, you'd better make that page. Add a new item to your site and call it *CreateAccount.aspx*. Add a `CreateUserWizard` control to the page, and that's most of the work done for you. Set the `ContinueDestinationPage` property to take users to *CustomerInfo.aspx*, as that's where they're most likely to want to go. In a slightly fancier site, you might take them to a confirmation page, and let them choose where to go from there, but for now, this will do.

Run your site and create a few users. Unfortunately, now that users are logged in, they can see the link to the Guard Schedule page. You could set the `Visibility` attribute of `hlGuardPage` to `false`, and add a `Page_Load` handler to *Welcome.aspx* to hide the link if users aren't a member of the Day shift or Night shift roles:

```
If (User.IsInRole("Day shift") Or User.IsInRole("Night shift")) Then
    hlGuardPage.Visible = True
    Else
        hlGuardPage.Visible = False
End If
```

Be sure to take the `HyperLink` out of the `LoginView` control and place it on the page, or the event handler won't be able to find it. That's certainly a good start. A logged-in user who's not a member of one of the guard roles can still type in the URL for the guard schedule page intentionally, and they'll be able to access the page, although they won't see the guard schedules. You can also make sure that users without privileges can't even access the guard page. First, create a page, *NoPrivs.aspx*, that has a polite message and a link back to *Welcome.aspx*. Now, add a bit of code to the `Page_Load` handler for *GuardPage.aspx*:

```
If Not (User.IsInRole("Day shift") Or User.IsInRole("Night shift")) Then
    Response.Redirect("NoPrivs.aspx")
End If
```

Now, any user that hasn't specifically been assigned to one of the two guard roles will be redirected to the *NoPrivs.aspx* page.

Exercise 9-3. This part isn't actually too difficult. Start in the WAT: create a new role, "Managers," and then add the two users mrand and dknight, adding them each to the Managers role. Next, add a page *Managers.aspx* to your site. Now you need to make sure only the managers can access this page. It's pretty easy to restrict this page to just users in the Manager role—just add the following to the `Page_Load` handler for *Managers.aspx*:

```
If Not (User.IsInRole("Manager")) Then
    Response.Redirect("NoPrivs.aspx")
End If
```

That keeps people who aren't managers out of the managers' page. Now, go back to *Welcome.aspx* and add a `HyperLink` pointing to *Managers.aspx*, but set its `Visible` property to `false`. You'll want to make sure the `HyperLink` is only visible to users with the Manager role, but you also want managers to be able to see the Guard schedule link, so add the following code to the `Page_Load` for *Welcome.aspx*:

```
If (User.IsInRole("Manager")) Then
    hlGuardPage.Visible = True
    hlManagers.Visible = True
Else
    hlManagers.Visible = False
End If
```

At the moment, managers can see the link to *GuardPage.aspx*, but if they try to access it, they'll get sent to *NoPrivs.aspx*. That's easy to fix. Open up *GuardPage.aspx.vb*, and add `User.IsInRole("Manager")` to the code that allows users to view the page, like this:

```
If Not (User.IsInRole("Day shift") Or User.IsInRole("Night shift")
        Or User.IsInRole("Manager")) Then
    Response.Redirect("NoPrivs.aspx")
End If
```

You also want managers to be able to see the schedule for all guards, and at the moment, they won't see any. There are a few ways to give managers permission to see the schedules; this is one of them:

```
If User.IsInRole("Day shift") Then
    pnlDayShift.Visible = True
    pnlNightShift.Visible = False
ElseIf User.IsInRole("Night shift") Then
    pnlNightShift.Visible = True
    pnlDayShift.Visible = True
ElseIf User.IsInRole("Manager") Then
    pnlDayShift.Visible = True
    pnlNightShift.Visible = True
End If
```

The `ElseIf` structure is starting to get a little awkward, and if you're going to add more roles, you should consider changing it to a `Select Case` statement, but for now, it's fine.

Now you have a site where the clients can access the promotional material, but nothing else; guards can access only their own schedules; and managers can access everything.

Chapter 10: Personalization

Answers to Quiz Questions

1. To enable user profiles, you need to add a line to *web.config*, setting `profile enabled` to `true`.
2. To specify the information to retain, you also need to add a `<properties>` section within the `profile` section, and use the `<add>` property to specify the names of the properties you want to retain.
3. The `Profile.IsAnonymous` property lets you know whether users are logged in.
4. You can also use the `<add>` property to save user information of types other than string. Simply specify the `type` in the `<add>` property in *web.config*, and specify the type, or the collection type.
5. To allow profile data for a user who's not logged in, use an anonymous profile by setting `<anonymousIdentification enabled="true">` in the *web.config* file.
6. The specific information that you want to retain for anonymous users should be marked with `allowAnonymous="true"` in the `<add>` property in the *web.config* file.
7. To migrate the user's profile data from an anonymous profile to the user's specific profile, use a global handler in the *global.asax* file to copy the data to the user's profile.
8. Style sheet themes are functionally equivalent to CSS style sheets, and can be overridden by the page or by the control. Customization themes are applied last, and therefore cannot be overridden.
9. You define the settings for a skin in a *.skin* file, which resides inside a folder named after the theme, which in turn is located with the `App_Themes` folder of your site.
10. To specify a theme for a specific page, add a `Theme` attribute to the `Page` directive for that page.

Answers to Exercises

Exercise 10-1. Create your new web site. Start off by deleting *Default.aspx*. Add a new page *Home.aspx*, set it as the start page, give it a title and an `<h1>`, and then add a `LoginStatus` control. Add a `LoginView` control, and set the Anonymous and Logged In templates to the appropriate messages. Add a `LoginName` control to the Logged In template to greet the user by name.

If you're not going to use the WAT to create any users, you need to add the following line to your *web.config* file, in the `<system.web>` section:

```
<authentication mode="Forms" />
```

Add a new page to your site, *login.aspx*. Give it a title and an `<h1>`, and then drag a Login control onto the page. Give it whatever formatting you like.

At the moment, though, the only way to create a new user is with the WAT. Add a page where users can create their own user accounts. Create a new page, *CreateAccount.aspx*. Give it a title and an `<h1>`, and then add a `CreateUserWizard` control to the page. Set the `ContinueDestinationPageUrl` property to *Home.aspx* to take users back to the front of the site when they've created their accounts. Go back to the *Login.aspx* page, and add a `Hyperlink` control to the Anonymous template of the `LoginView` control, with a `NavigateUrl` property of `CreateAccount.aspx`, so that users can create an account if they don't already have one.

Run your site and create a handful of users to populate your database. Be sure to write down the passwords because you'll need them later.

To enable profiles, the first thing you need to do is make a modification to the *web.config* file. Open *web.config*, and add the following code to the `<system.web>` section to enable profiles and store the four data elements you want to save:

```
<profile enabled="true" defaultProvider="AspNetSqlProfileProvider">
  <properties>
    <add name="userName" />
    <add name="numFish" />
    <add name="fishType" />
    <add name="favFish" />
  </properties>
</profile>
```

Next, add a new page, *ProfilePage.aspx*, to the site. Use a table for layout, and enter the standard controls as shown in the figure. Give the controls meaningful names so that you can access them from the event handler. Be sure to add the Save button at the bottom of the table.

Double-click the Save button to create a handler for the `Click` event. You need to record the string values from the form into the `Profile` object.

```
Protected Sub btnSave_Click(ByVal sender As Object, ByVal e _
                  As System.EventArgs) Handles btnSave.Click
    If Profile.IsAnonymous = False Then
        Profile.userName = txtUserName.Text
        Profile.numFish = txtNumFish.Text
        Profile.fishType = rblFreshTropical.SelectedValue
        Profile.favFish = ddlFavFish.SelectedItem.Text
    End If
    Response.Redirect("Home.aspx")
End Sub
```

You have to write a little bit of code to extract the text values from the radio button list, and the drop-down list, and then enter them into the `Profile` attributes.

You'll also need to create the `Page_Load` event handler for the profile page, so that if the user already has a profile, those choices will be pre-selected when the user comes to this page to edit her profile. That handler should look like this:

```
If Not IsPostBack And Profile.userName IsNot Nothing Then
    If Profile.IsAnonymous = False Then
        txtUserName.Text = Profile.userName
        txtNumFish.Text = Profile.numFish
        rblFreshTropical.SelectedValue = Profile.fishType
        ddlFavFish.SelectedValue = Profile.favFish
    End If
End If
```

Notice that the Save button handler redirects to *Home.aspx* when it's done. The next thing you need to do is make some modifications to *Home.aspx* so that users can see their profiles. First, switch the `LoginView` control to the logged-in template, and add a hyperlink directing users to the Profile page.

Now, add a panel, `pnlProfileInfo`, so that users can see the current contents of their profile. The markup for this panel and its contents should look like this:

```
<asp:Panel ID="pnlProfileInfo" runat="server" Visible="False" Width="250px">
    <table>
        <tr>
            <td>User Name:</td>
            <td>
                <asp:Label ID="lblName" runat="server"></asp:Label>
            </td>
        </tr>
        <tr>
            <td>Number of fish:</td>
            <td>
                <asp:Label ID="lblNumFish" runat="server"></asp:Label>
            </td>
        </tr>
        <tr>
            <td>Tropical or Fresh?</td>
            <td>
                <asp:Label ID="lblFishType" runat="server"></asp:Label>
            </td>
        </tr>
        <tr>
            <td>Favorite Fish</td>
            <td>
                <asp:Label ID="lblFavFish" runat="server"></asp:Label>
            </td>
        </tr>
    </table>
</asp:Panel>
```

Now you just have to write an event handler to populate the labels in the table in the panel. Add the following code to the Page_Load event handler:

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles Me.Load
    If Not IsPostBack And _
            Profile.UserName IsNot Nothing And _
            Profile.IsAnonymous = False Then
        pnlProfileInfo.Visible = True
        lblName.Text = Profile.userName
        lblNumFish.Text = Profile.numFish
        lblFishType.Text = Profile.fishType
        lblFavFish.Text = Profile.favFish
    Else
        pnlProfileInfo.Visible = False
    End If

End Sub
```

Exercise 10-2. Start by copying Exercise 10-1 to a new web site, Excercise 10-2, and be sure to set *Home.aspx* as the Start Page. Add the following line to *web.config*, inside the <system.web> section:

```
<anonymousIdentification enabled="true" />
```

In the <profile> section, add allowAnonymous="true" to both the fishType and favFish elements, so that the anonymous profile can use them.

Next, make some changes to *ProfilePage.aspx*. Break the existing table into two so that the "name" and "number of fish" controls are separated from the "fish type" and "favorite fish" controls. Add a Panel control around the name and number of fish controls because those are the controls that should be invisible to anonymous users.

Now you need to make two changes to the Page_Load event handler. First, you need to make sure that pnlRegisteredOnly is hidden from anonymous users. Second, if there's any profile information in the fishType or favFish properties, even if the profile is anonymous, you want to set the controls accordingly. So, make the following changes to the Page_Load event handler:

```
If Not IsPostBack And Profile.userName IsNot Nothing Then
    If Profile.IsAnonymous = True Then
        pnlRegisteredOnly.Visible = False
    Else
        pnlRegisteredOnly.Visible = True
        txtUserName.Text = Profile.userName
        txtNumFish.Text = Profile.numFish
    End If
    If Profile.fishType IsNot Nothing Then
        rblFreshTropical.SelectedValue = Profile.fishType
    End If
    If Profile.favFish IsNot Nothing Then
        ddlFavFish.SelectedValue = Profile.favFish
```

```
        End If
    End If
```

Next, you need to make changes to the `btnSave_Click` handler. All you really need to do in this case is move the profile assignment statements for `fishType` and `favFish` outside the `If` block that checks whether the user is anonymous, like this:

```
If Profile.IsAnonymous = False Then
    Profile.userName = txtUserName.Text
    Profile.numFish = txtNumFish.Text
End If
Profile.fishType = rblFreshTropical.SelectedValue
Profile.favFish = ddlFavFish.SelectedItem.Text
```

Now, go back to *Home.aspx* and make some changes there so that anonymous users can access the profile page. Break the table that displays the profile information into two; leave the first two rows inside `pnlProfileInfo`, and move the last two rows outside (this is easy to do if you move the `</asp:Panel>` tag in Source mode, and then fix the `<table>` tags accordingly). In addition, take the hyperlink that points to the profile page out of the `LoginView` control, and place it on the page above the panel. That takes care of the *.aspx* file, but you still have to fix the code-behind file.

Open up *Home.aspx.vb*. You'll need to modify the `Page_Load` handler so that the `fishType` and `favFish` profile information is displayed for anonymous users, but only if it exists. Change the handler to look like this:

```
If Not IsPostBack And _
Profile.userName IsNot Nothing Then
    If Profile.IsAnonymous = False Then
        pnlProfileInfo.Visible = True
        lblName.Text = Profile.userName
        lblNumFish.Text = Profile.numFish
    Else
        pnlProfileInfo.Visible = False
    End If
End If
If Profile.fishType IsNot Nothing Then
    lblFishType.Text = Profile.fishType
End If
If Profile.favFish IsNot Nothing Then
    lblFavFish.Text = Profile.favFish
End If
```

The next step is to make it possible to migrate anonymous data to the user's new profile. For that, you need to create a global event handler. So, click Website → Add New Item, select Global Application Class, and accept the default filename. Now you just need to add an event handler that assigns the fishType and favFish properties of the anonymous profile to the new profile you just created:

```
Sub Profile_MigrateAnonymous(ByVal sender As Object, ByVal e As
ProfileMigrateEventArgs)
    Dim anonymousProfile As ProfileCommon = Profile.GetProfile(e.AnonymousID)
    If anonymousProfile IsNot Nothing Then
```

```
            If anonymousProfile.fishType IsNot Nothing Then
                Profile.fishType = anonymousProfile.fishType
            End If
            If anonymousProfile.favFish IsNot Nothing Then
                Profile.favFish = anonymousProfile.favFish
            End If
        End If
    End Sub
```

There you go. Your site can now accommodate anonymous users and migrate their data to a new profile.

Exercise 10-3. Start by copying Exercise 10-2 to a new site, Exercise 10-3, and be sure to set *Home.aspx* as the Start Page. Next, create the *App_Themes* folder in Solution Explorer by right-clicking the root of the site and choosing Add ASP.NET Folder → Theme. Now, add the two theme folders for Angelfish and Clownfish within the *App_Themes* folder. Create the *Button.skin* and *Label.skin* files in the Angelfish folder. Here's the markup for the *Button.skin* file in the Angelfish theme:

```
<asp:Button runat="server"
    ForeColor="Yellow"
    BackColor="Black"
    Font-Size="Large" />
```

The other skin files are similar, but with different colors and names, of course.

Next, you have to enable themes in the `properties` subelement of the `profile` section in *web.config*:

```
<add name="Theme" />
```

Now, open up *Home.aspx.vb* and add the `overrides StyleSheetTheme( )` method as follows:

```
Public Overrides Property StyleSheetTheme( ) As String
    Get
        If Profile.IsAnonymous = False And Profile.Theme IsNot Nothing Then
            Return Profile.Theme
            Else
                Return "Angelfish"
        End If
    End Get
    Set(ByVal value As String)
        Profile.Theme = value
    End Set
End Property
```

When the user logs in, the theme stored in the user's profile will be loaded. The anonymous users will get the Angelfish theme.

Next, you need to provide a way for users to choose their theme. Add two buttons to *Home.aspx*, inside the panel `pnlProfileInfo`—one labeled Angelfish Theme and the other Clownfish Theme, with IDs of `btnAngelfish` and `btnClownfish`, respectively.

Create an event handler to use for the `Click` event for both buttons. Call it `Theme_Click` and add the highlighted code from the following snippet:

```
Protected Sub Theme_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
            Handles btnClownfish.Click, btnAngelFish.Click
    Dim btn As Button = CType(sender, Button)
    If btn.Text = "Clownfish Theme" Then
        Profile.Theme = "Clownfish"
    Else
        Profile.Theme = "Angelfish"
    End If
    Server.Transfer(Request.FilePath)
End Sub
```

If the button that raises the `Click` event has the text "Clownfish Theme," the Clownfish theme is set. Otherwise, the Angelfish theme is set.

Finally, create an event handler for the `PreInit` event and add the highlighted code from the following snippet:

```
Protected Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs) _
        Handles Me.PreInit
    If Profile.IsAnonymous = False Then
        Page.Theme = Profile.Theme
    End If
End Sub
```

Now, when you run the app, the default theme on the home page will be Angelfish, and if you log in and change the theme for user, that theme will be remembered the next time the user logs in.

Index

Symbols

&= (concatenate assignment operator), 278
-= (decrement assignment operator), 278
/= (divide assignment operator), 278
= (equal to operator), 31
> (greater than operator), 31
>= (greater than or equal to operator), 31
+= (increment assignment operator), 278
< (less than operator), 31
<= (less than or equal to operator), 31
_ (line continuation character), 12
% (modulus operator), 200
*= (multiply assignment operator), 278
<> (not equal to operator), 31
? (question mark), prepending expressions in debugger, 311
[] (square brackets), in SELECT statement, 122

A

access modifiers, Visual Basic, 236
access to pages, restricting, 342–348
Accordion control, 84
Add Application dialog box, 498
Add Connection dialog box, 331, 458
Add New Item dialog box, 85, 148, 241
Add ORDER BY Clause dialog box, 143
Add Style Rule dialog box, 214
Add WHERE Clause dialog box, 142, 144
AddHistoryPoint method, 80
Adobe Classroom in a Book, 39
Advanced SQL Generation Options dialog box, 130
AdventureWorks application
  basic controls in, 36–38
  collapsible panels in, 97–100
  images in, 56
  links in, 57
  panels in, 45
  partial-page updates for, 72–76
  pop-up windows in, 90–95
  selection controls in, 43, 45–54
  source code for, 58–63
  tables in, 39–40
  watermarks for, 85–89
AdventureWorks database, 112, 458–460
AJAX (Asynchronous JavaScript and XML), 19, 20, 22, 69–71, 414
  browser history not affected by, 77
  browser history, enabling, 77–83
  compared to Silverlight, 34
  enabling, 24
  history of, 70
  technologies used by, 70
AJAX Control Toolkit, 22, 83
  controls in, list of, 84
  installing, 464
aliases, in queries, 416
All Together application, 388
  AJAX in, 414
  Cart page, 414–419, 430
  configuration for, 447
  Confirm page, 426–428, 433
  error pages for, 428
  Login page, 398–402, 435
  logo for, 389
  master pages in, 392–395, 436

We'd like to hear your suggestions for improving our indexes. Send email to *index@oreilly.com*.

All Together application *(continued)*
navigation for, 402
Products page, 403–414, 438
Purchase page, 419–426, 442
roles for, 396–397
source code for, 430–451
styles for, 389–392
users for, 396–397
AllowPaging property, GridView control, 126
AllowSorting property, GridView control, 126
AlwaysVisibleControlExtender control, 84
anonymous personalization, 364–371
adding profile for, 365–370
enabling, 365
migrating anonymous data to a user, 370
App_Themes folder, 374
appearance (design) of web sites, 39
application examples
AdventureWorks (see AdventureWorks application)
All Together (see All Together application)
AspNetDocumentLevelStyles, 210
AspNetInLineStyles, 208–210
AWProductData, 115, 166–172
ButtonNavigation, 236–239
ChangingTheMasterPage, 232–234
CompareValidator, 189–195
CustomErrorPage, 318
CustomValidator, 198–200
DebuggingApp, 292–295
DynamicAdventureWorks, 162–165
FormsBasedSecurityWAT, 327
Hello World, 2, 6–13
LifeCycle, 263–267
LinqDataSource, 148–157
LinqDataSourceEditing, 157–160
MasterPages, 221–227
NestedMasterPages, 228–231
Postbacks, 24–34
RangeValidator, 195
RegularExpressionValidator, 197
RequiredFieldValidator, 182–188
RequiredFieldValidatorSummary, 188
SecurityRoles, 340
SessionState, 280–285
SitePersonalization, 353–364
SitePersonalizationComplex, 360
source code for, 166–172
StateBag, 275–279
Themes, 372
ViewState, 270–275
web site for, 388
WebSiteNavigation, 241–243
Application keyword, 285
application state, 262, 268, 285
applications, web site (see web site applications)
application-wide error pages, 316–319
arrays, Visual Basic, 276
ASP.NET AJAX controls (see AJAX)
ASP.NET Dynamic Data, 161–165
ASP.NET server controls (see server controls)
ASPNETDB.MDF file, 331
AspNetDocumentLevelStyles application, 210
AspNetInLineStyles application, 208–210
.aspx files (see markup files)
assignment operators, Visual Basic, 278
Asynchronous JavaScript and XML (see AJAX)
asynchronous postbacks, 23
creating, 29–34
not included in browser history, 77
authentication
forms-based, 326, 327, 331, 396
Windows authentication, 118, 327
AutoGenerateColumns property, GridView control, 126
automatic renaming, 37
AutoPostBack property
CheckBox control, 47, 56
RadioButton control, 72, 74
AWProductData application, 115, 166–172

B

books
Classroom in a Book (Adobe), 39
Creative Suite 3 (Total Training course), 39
CSS: The Definitive Guide (Meyer), 208
The Design of Everyday Things (Norman), 45
Don't Make Me Think (Krug), 39, 45
Essential Silverlight (Wenz), 34
HTML & XHTML: The Definitive Guide (Musciano; Kennedy), 19, 208
Mastering Regular Expressions (Friedl), 196
The Non-Designer's Design Book (Williams), 39

Programming ASP.NET (Liberty; Hurwitz), 20, 112, 348
Programming ASP.NET 3.5 (Liberty; Hurwitz; Maharry), xii
Programming C# 3.0, Fifth Edition (Liberty; Xie), 147
Programming .NET 3.5 (Horovitz; Liberty), 147
Programming Silverlight (Liberty; Heuer), 34
Programming Visual Basic 2008 (Patrick), 147
Transact-SQL Programming (Kline et al.), 412
The Zen of CSS Design: Visual Enlightenment for the Web (Shea; Holzschlag), 39
Boolean variables, 28
bread crumbs, 240–243, 252–254
Break All option, Debug menu, 302
Breakpoint Condition dialog box, 306
Breakpoint Hit Count dialog box, 307
Breakpoint window, 303
breakpoints, 29, 303–308
deleting, 305
enabling and disabling, 303, 304, 305
evaluating expressions at, 305
hit count for, 306
icons for, 307
performing actions at, 307
setting, 303, 305
stepping through code using, 309
viewing all breakpoints, 303
(see also symbolic debugging)
browser history, 76–83
BulletedList control, 43
BulletStyle property, BulletedList control, 43
Button control, 9, 235–239
ButtonNavigation application, 236–239

C

C#
compared to Visual Basic, xiii
examples in, web site for, xiii
this keyword, 237
C# code-behind files (.cs files), 19
Calendar control, 84
Call Stack window, for debugging, 311, 313
Cascading Style Sheets (see CSS)
CascadingDropDown control, 84
catch blocks, 291
CausesValidation property, postback controls, 180
ChangingTheMasterPage application, 232–234
CheckBox control, 43, 138
CheckBoxList control, 43, 46–47
CheckChanged control, 74
Checked property
CheckBox control, 138
ListItem object, 54
TextBox control, 32
CheckedChanged event
CheckBox control, 56
RadioButton control, 74
ChildNodes property, SiteMapNode object, 251
class property, 218
classes, 21
classes, style, 213
Classroom in a Book (Adobe), 39
Click event, Button control, 11, 28, 238
client technologies, 69
(see also AJAX)
ClientValidationFunction property, CustomValidator control, 199
code examples
in C#, web site for, xiii
permission to use, xv
code-behind files, 7, 11, 19, 22
(see also event handlers)
CollapseControlID property, CollapsiblePanelExtender control, 98
Collapsed property, CollapsiblePanelExtender control, 98
CollapsedImage property, CollapsiblePanelExtender control, 99
CollapsedText property, CollapsiblePanelExtender control, 99
CollapsiblePanelExtender control, 84, 95–100
colors in style sheets, specifying, 216
columns, database, 114
CompareValidator application, 189–195
CompareValidator control, 181, 189–195
concatenate assignment (&=) operator, 278
concatenation of strings, 55
conditional operators, 31

configuration file (web.config)
AppSettings in, to preserve state, 285
authentication mode, setting, 331
connection strings, saving in, 119, 271
debugging, enabling, 9, 301
error handling, configuring, 316, 318, 428
profiles, configuring, 354
session timeout, setting, 279
view state, enabling or disabling, 269
Configure Data Source Wizard, 116–123, 129
Configure Select Statement dialog box, 129, 143
Configure Select Statement Wizard, 144
ConfirmButtonExtender control, 84
connection strings, 118–120
constraints, 132
Content control, 229
content files (see markup files)
ContentPlaceHolder control, 222, 226, 229
ContentPlaceHolderID property, Content control, 229
Continue option, Debug menu, 302
control state, 268
control tree, 263, 264
controls, 7, 19
adding, 8–13
AJAX Control Toolkit, 22, 83, 84, 464
AJAX controls (see AJAX)
custom controls, 20
data source controls, 20, 113–121
data view controls, 20, 112
events for, displaying, 35
extenders (see extenders)
HTML controls, 19
ID property for, 35
login controls, 20
personalization controls, 20
properties of (see properties, of controls)
rich controls, 21
security controls, 20
selection controls, 43, 50–54
server controls, 18, 19, 20–22
types of, 19–22
user controls, 20
validation controls, 20, 180
view state for, disabling, 270
(see also specific controls)
ControlToValidate property
CustomValidator control, 199
RangeValidator control, 196
RegularExpressionValidator control, 197
RequiredFieldValidator control, 184
Copy Web Site dialog box, 482
CreateUserWizard control, 334
Creative Suite 3 (Total Training course), 39
CRUD (Create, Read, Update, Delete) database operations, 111
Dynamic Data for, 161
generating for DataSource, 128–133
.cs files (C# code-behind files), 19
CSS (Cascading Style Sheets), 42, 208
(see also style sheets)
CSS: The Definitive Guide (Meyer), 208
CssClass property, all controls, 213, 218
CType method, 139
CurrentNode property, SiteMap object, 251
custom controls, 20
CustomErrorPage application, 318
<customErrors> element, 316–319
customization themes, 371, 379
CustomValidator application, 198–200
CustomValidator control, 181, 198–200

D

data source controls, 20, 113–121
Data tab, 115
data view controls, 20, 112
data, persistent, 111
data, validating (see validation of data)
database
binding to data source controls, 113–121
connection string for (see connection strings)
constraints in, 132
CRUD operations with, 111
relational, 114
support for, with hosting company, 479
uploading to published web site, 482–493
Database Publishing Wizard, 483
DataContext object, 148, 150
DataList control, 112
DataSource controls, 115–123
DataSource property, data view controls, 114
.dbml file, 152
Debug menu, 302
Debug toolbar, 302
debugging (see symbolic debugging; tracing)
Debugging Not Enabled dialog box, 10
DebuggingApp application, 292–295
decrement assignment (-=) operator, 278

defaultRedirect attribute, <customErrors> element, 317
DELETE statement, generating for DataSource, 128–133
DeleteCommand control, 132
The Design of Everyday Things (Norman), 45
design of web sites, 39
design surface, 7
Design tab, 7
Design view, 7
DetailsView control, 112, 410
dictionaries, Visual Basic, 276
directives
  Master directive, 229
  MasterType directive, 233, 394
  Page directive, 18
  (see also specific directive attributes)
DiscountASP.NET, 481
Display property, RequiredFieldValidator control, 184, 186
DisplayMode property, ValidationSummary control, 189
 element, 218
divide assignment (/=) operator, 278
dll (dynamic link library), 150
document level styles, 208, 210
DOM (Document Object Model), AJAX using, 70
domain name, 479, 480, 500
Don't Make Me Think (Krug), 39, 45
dot notation, 21
DragPanelExtender control, 84
DropDownList control, 43, 49, 50
DropShadow control, 84
Dynamic Data, 161–165
dynamic link library (dll), 150
DynamicAdventureWorks application, 162–165

E

email forwarding addresses, 479
EnableClientScript property, CustomValidator control, 199
EnableHistory property, ScriptManager control, 77
EnablePartialRendering property, ScriptManager control, 71, 75
EnableSessionState attribute, Page directive, 284
EnableViewState attribute, Page directive, 269, 273, 274
EnableViewState property
  for application, 269
  for controls, 270
entities, LINQ object model, 148
equal to operator (=), 31
error handling, 291, 313–320
  application wide, 428
  logic errors, 314
  syntax errors, 314
  unhandled errors, results of, 314
error page
  application-wide, 316–319
  default, 315
  page-specific, 320
ErrorMessage property, RequiredFieldValidator control, 184, 189
ErrorPage attribute, Page directive, 320
errors, catching, 252
Essential Silverlight (Wenz), 34
event handlers, 11, 22, 30
  arguments of, 30
  creating, 11–13, 74
  linking to event, 30
  with postbacks, 23
event-driven web site applications, 11
events, 11, 22
  displaying for controls, 35
  linking to event handlers, 30
examples (see application examples; code examples)
ExpandControlID property, CollapsiblePanelExtender control, 98
ExpandDirection property, CollapsiblePanelExtender control, 99
ExpandedImage property, CollapsiblePanelExtender control, 100
ExpandedText property, CollapsiblePanelExtender control, 100
explicitly named skins, 372, 381
expressions, evaluating in debugger, 311
Extender Wizard dialog box, 87
extenders, 85
  CollapsiblePanelExtender control, 95–100
  PopupControlExtender control, 89–95
  TextBoxWaterMarkExtender control, 84–89

external styles, 208
(see also style sheets)

F

Fields dialog box, 127, 133, 409
File Breakpoint dialog box, 306
File System location option, 3, 4
files for web site
accessing, 4
specifying location of, 3, 5
files, source (see code-behind files; markup files)
FilteredTextBoxExtender control, 84
Find dialog, 47
firewall, configuring, 496
For Each loop, Visual Basic, 251
For loop, Visual Basic, 295
ForeColor property, table cells, 138
foreign keys, 114
forms-based security, 326
database for, 331
setting, 327, 396
FormsBasedSecurityWAT application, 327
FormView control, 112
forwarding addresses, 479
Friedl, Jeffrey, (Mastering Regular Expressions), 196
FTP location option, 5
FTP Log On dialog box, 5
functions
methods implemented as, 30
on call stack, examining during debugging, 313
stepping out of, during debugging, 310
stepping over, during debugging, 310

G

Garrett, Jesse James (AJAX coined by), 70
Get method, 378
global.asax file, 162, 370
Globally Unique Identifier (GUID), 367
graphics (images), 56
greater than operator (>), 31
greater than or equal to operator (>=), 31
GridView control, 112, 407, 418
adding to page, 123–125
auto-generated code for, 125–127
binding to database tables, 113–121
commands in, modifying appearance of, 133
contents of, modifying conditionally, 135–138
in UpdatePanel, 125
LINQ object model used with, 153–160
paging for, enabling, 124, 157
selecting data from, 139
sorting for, enabling, 124, 157
specifying DataSource for, 124
testing, 134
GroupName property, RadioButton control, 44
GUID (Globally Unique Identifier), 367

H

Handles keyword, 30
hardware requirements, 453
HasChildNodes property, SiteMapNode object, 251
hashtables, Visual Basic, 276
Hello World application, 2
controls for, 8–13
creating, 6–8
helper methods, 273
Heuer, Tim (Programming Silverlight), 34
hidden fields, view state saved in, 269
history points, 77
history, browser (see browser history)
Holzschlag, Molly (The Zen of CSS Design: Visual Enlightenment for the Web), 39
Horovitz, Alex (Programming .NET 3.5), 147
hosting web sites (see publishing web sites)
HoverMenuExtender control, 84
HTML & XHTML: The Definitive Guide (Musciano; Kennedy), 19, 208
HTML (HyperText Markup Language), 1, 17
(see also markup files)
HTML controls, 19
HTTP location option, 5
HTTP status codes, 296
HttpResponse class, 239
Hungarian notation, 26
Hurwitz, Dan
Programming ASP.NET, 20, 112, 348
Programming ASP.NET 3.5, xii
HyperLink control, 57, 235–239
hyperlinks, 57

I

ID property, all controls, 35
IDE (Integrated Development Environment), 2
If statements, 137
If-Then statements, 31
IIS, publishing web site to, 498–499
Image control, 56, 97
ImageButton control, 56
ImageControlID property, CollapsiblePanelExtender control, 99
ImageMap control, 56
images, 56
ImageURL property, Image control, 56, 97
Immediate window, for debugging, 311
impedance mismatch, 146
@import statement, 216
increment assignment (+=) operator, 278
Init event, Page, 265
InitialValue property, RequiredFieldValidator control, 184
inline styles, 208–210
inner HTML, 52
inner join, 412
INSERT statement, generating for DataSource, 128–133
Insert Table dialog box, 39
Insert Table Wizard, 39, 52
InsertCommand control, 133
instances, 21
Integrated Development Environment (IDE), 2
IntelliSense, finding properties with, 35
IP address, 480
IsPostBack property, Page, 23, 265
IsValid property, Page, 186, 265
Item editor, adding list items using, 46–47

J

JavaScript
  AJAX using, 70
  custom validators using, 199
  modulus operator (%), 200
  script blocks written in, 18
  writing AJAX controls in, 71
joins, SQL, 412, 415
JScript, RegularExpressionValidator control using, 197

K

Kennedy, Bill (HTML & XHTML: The Definitive Guide), 19, 208
Kline, Kevin (Transact-SQL Programming), 412
Krug, Steve (Don't Make Me Think), 39, 45

L

Label control, 9
Language Integrated Query (see LINQ)
.layout file, 153
layout, methods of, 42
less than operator (<), 31
less than or equal to operator (<=), 31
Liberty, Jesse
  Programming ASP.NET, 20, 112, 348
  Programming ASP.NET 3.5, xii
  Programming C# 3.0, Fifth Edition, 147
  Programming .NET 3.5, 147
  Programming Silverlight, 34
life cycle of page, 261–267
LifeCycle application, 263–267
line continuation character (_), 12
link tag, 217
LinkButton control, 58, 235–239
links (see hyperlinks)
LINQ (Language Integrated Query), 146–147
  editing data, 157–160
  object model
    creating, 147–153
    templates used with, 162
    using with GridView, 153–160
    with Dynamic Data, 162
LinqDataSource application, 148–157
LinqDataSource control, 148, 153, 157
LinqDataSourceEditing application, 157–160
list selection controls, 45–49
ListBox control, 43, 50
ListItem Collection Editor dialog box, 360
ListItem object, 46
ListView control, 112
LiteralControls, 264
literals, in regular expressions, 196
Load event, Page, 27–29, 265
Load phase, of page life cycle, 267
localhost, 10, 343
Locals window, for debugging, 311, 312

Location for web site
File System option, 3, 4
FTP option, 5
HTTP option, 5
logging in
determining if user is logged in, 336, 344–345
enabling for users, 329
forms-based security for, 326, 327, 331, 396
Login page, 337–340, 398–402
of previously anonymous user, migrating data to, 370
logic errors, 314
login controls, 20
Login page, 337–340, 398–402
LoginView control, 336
loops
For Each loop, Visual Basic, 251
For loop, Visual Basic, 295

M

Maharry, Dan (Programming ASP.NET 3.5), xii
MaintainScrollPositionOnPostback property, Page directive, 75
markup files (.aspx files), 6, 7, 17
Master directive, 229
.master files, 222
master pages, 21, 207, 220
changing at runtime, 232–234
compared to style sheets, 220
content pages for, 225–227
creating, 221–224, 392–395
for navigation features, 235
nesting, 228–231
public property for, 232
Mastering Regular Expressions (Friedl), 196
MasterPageFile attribute, Master directive, 229
MasterPages application, 221–227
MasterType directive, 233, 394
MaximumValue property, RangeValidator control, 196
Me keyword, Visual Basic, 237
measurement units, in style sheets, 216
Menu control, 248
menus, for navigation, 240–243
metacharacters, in regular expressions, 196
methods, 21
as functions, 30
on call stack, examining during debugging, 313
passing values to, 30
stepping out of, during debugging, 310
stepping over, during debugging, 310
as subs, 30
Meyer, Eric (CSS: The Definitive Guide), 208
MinimumValue property, RangeValidator control, 196
Mod operator, 200
mode attribute, <customErrors> element, 317
Modify Style dialog box, 214
modulus operator (%), 200
multiply assignment (*=) operator, 278
Musciano, Chuck (HTML & XHTML: The Definitive Guide), 19, 208
MutuallyExclusiveCheckBoxExtender control, 84

N

named skins, 372, 381
namespaces, 25
naming conventions, 26
navigation
bread crumbs for, 240–243, 252–254
Buttons for, 235–239
HyperLinks for, 235–239
LinkButtons for, 235–239
master pages for, 235
menus for, 240–243
site maps for, 243–252, 402
nested master pages, 228–231
NestedMasterPages application, 228–231
.NET AJAX controls (see AJAX)
.NET, technologies in, 34
Network Solutions, 479
New Web Site dialog box, 4, 162, 469
NoBot control, 84
The Non-Designer's Design Book (Williams), 39
normalized data, 412
Norman, Donald A. (The Design of Everyday Things), 45
not equal to operator (<>), 31

O

object model, LINQ
  creating, 147–153
  templates used with, 162
  using with GridView, 153–160
  with Dynamic Data, 162
Object Relational Designer, 147–153, 162
object-oriented terminology, 21
objects, 21
Open Web Site dialog box, 472, 475, 482
operating system requirements, 453
Operator property, CompareValidator control, 192, 193, 195
operators
  assignment, 278
  conditional, 31
Options dialog box, 470, 471
O/R (Object Relational) Designer, 147–153, 162
order form, example of (see AdventureWorks application)
OrderForm markup, 63
outer join, 412
Output option, Debug menu, 303

P

Page class, 263
Page directive, 18
  (see also specific attributes of Page directive)
Page Initialization phase, of page life cycle, 265
page layout, methods of, 42
Page_Load method, 265
Page_PreRender method, 265
page-level tracing, 296–298
pages, 17–19
  life cycle of, 261–267
  restricting access to, 342–348
  (see also web site applications)
page-specific error pages, 320
Panel control, 45, 95–100
parameters, SQL, 132, 140–145
partial page postbacks (see asynchronous postbacks)
partial-page updates, 71
passing parameters, 141
PasswordRecovery control, 340
passwords
  removing and setting a new one, 340
  strong passwords, 330
PasswordStrength control, 84
PathSeparator property, SiteMapPath control, 254
Patrick, Tim (Programming Visual Basic 2008), xii, 147
permissions (see roles)
persistent data, 111
personalization, 353
  anonymous, 364–371
  based on roles, 399–402
  forms-based security allowing, 326
  profiles for (see profiles)
personalization controls, 20
physical directories, mapping virtual directories to, 467–469
pinning windows, 8
pop-up windows, 89–95
PopupControlExtender control, 84, 89–95
PopupControlID property, PopupControlExtender control, 91
port forwarding, 497
port numbers, 343
Position property, PopupControlExtender control, 92, 95
postbacks, 22
  asynchronous, 23, 29–34, 77
  synchronous, 23, 27–29
Postbacks application, 24–34
PreInit event, Page, 265
PreRender event, Page, 265
PreviousPage property, Page, 265
primary key, 114
<profile> element, 354
Profile object, 357
profiles, 353–364
  checking for existence of, 357
  complex data types in, 359–364
  creating, 354–359
  for anonymous users, 365–370
Programming ASP.NET (Liberty; Hurwitz), 20, 112, 348
Programming ASP.NET 3.5 (Liberty; Hurwitz; Maharry), xii
Programming C# 3.0, Fifth Edition (Liberty; Xie), 147

Programming .NET 3.5 (Horovitz; Liberty), 147
Programming Silverlight (Liberty; Heuer), 34
Programming Visual Basic 2008 (Patrick), xii, 147
<properties> element, 354
Properties window, 7, 24, 35, 35–36
properties, of classes, 21
properties, of controls
  accessing programmatically, 378
  displaying, 7, 35
  examining during debugging, 311
  finding with IntelliSense, 35
  setting, 41
  (see also specific properties)
public property, master pages, 232
publishing web sites
  hosting company for, 478–480
    account setup, 481
    ASP.NET support from, 481
    choosing, 481
    database support from, 479
    domain name for, 479, 480
    testing and maintenance of site, 493
    uploading data to, 482–493
    uploading web site files to, 482
  hosting your own site, 494–501
    domain name for, 500
    security for, 495–497
    under IIS, 498–499
    web server software for, 495

Q

queries (see SELECT statement)
question mark (?), prepending expressions in debugger, 311

R

RadioButton control, 43, 74
RadioButtonList control, 43, 48, 405, 421
RangeValidator application, 195
RangeValidator control, 181, 195
ReadOnly property, TextBox control, 53, 139
records, database, 114
red dot, indicating breakpoint, 303
Redirect method, HttpResponse class, 239
RegEx Buddy program, 198
RegularExpressionValidator application, 197
RegularExpressionValidator control, 181, 196, 421, 422
relational database, 114
renaming, automatic, 37
Render event, Page, 265
Rendering phase, of page life cycle, 265
ReorderList control, 84
Repeater control, 112
RequiredFieldValidator application, 182–188
RequiredFieldValidator control, 181, 419
  creating, 182–188
  using with CompareValidator control, 192, 194
RequiredFieldValidatorSummary application, 188
Response propert, 265
Restart option, Debug menu, 302
rich controls, 21
roles, 340
  creating, 396–397
  customized content for, 399–402
  limiting page access based on, 345–348
RoundedCornersExtender control, 84
RowDataBound event, GridView control, 135–137
rows, database, 114

S

scaffolding, 162, 164
schema names for tables, 121–123
script blocks, 18
ScriptManager control, 24, 71–76, 77, 80, 414
security
  access to pages, restricting, 342–348
  database for, 331
  for web server, 495–497
  forms-based, 326, 327, 331, 396
  of view state, 269, 270
  (see also WAT)
security controls, 20
security database, 331
SecurityRoles application, 340
Select Case statement, Visual Basic, 283
SELECT statement, 114
  aliases in, 416
  specifying for DataSource control, 120–123
  square brackets ([]) in, 122
  (see also SQL)
Select Style Sheet dialog box, 218
Select URL dialog box, 334

Selected property, ListItem object, 47
SelectedIndex property, list controls, 52
SelectedIndexChanged event
  GridView control, 139
  RadioButtonList control, 281
SelectedItem property, list controls, 52, 54
SelectedValue property, list controls, 52, 81
selection controls, 43, 44, 50
  feedback on selections, providing, 52–54
  retrieving selections from, 52
  (see also list selection controls)
selectors, styles for, 213
<SelectParameters> element, SqlDataSource control, 145, 416
server (see web server)
server controls, 18, 19, 20–22
ServerValidate event, CustomValidator control, 199
session, 262, 279
Session keyword, 280
session state, 268, 279–285
session timeout, 279
Session_Start event, session, 280
SessionState application, 280–285
Set method, 378
SetFocusOnError property, RequiredFieldValidator control, 186
Shea, Dave (The Zen of CSS Design: Visual Enlightenment for the Web), 39
ShowStartingNode property, SiteMapDataSource control, 247
Silverlight, 34
site maps, 243–252
  accessing nodes programmatically, 249–252
  creating, 243–245, 402
  Menu used with, 248
  root node of, adjusting, 247
  TreeView used with, 246–248
<sitemap> element, 244
.sitemap files, 245
SiteMapDataSource control, 245, 247
<siteMapNode> element, 244
SiteMapNode object, 251
SiteMapNodeCollection object, 251
SiteMapPath control, 252
SitePersonalization application, 353–364
SitePersonalizationComplex application, 360
skins, 371
  creating, 374–375
  default skins, 372
  enabling, 375
  explicitly named skins, 372, 381
Slider control, 84
.sln files, 470
Smart Tags, 46, 404
software requirements, 453
Solution Explorer window, 7
solution file, 470
SortExpression property, GridView control, 126
source code (see code-behind files; code examples; markup files)
Source tab, 7
Source view, 7
  adding list items using, 47–49
  split with Design view, 7
 element, 218
Split tab, 7
spoofing, 180
SQL (Structured Query Language), 114
  joins, 412, 415
  passing parameters, 132, 140–145
  (see also SELECT statement)
SQL Server Express, configuring, 457
SqlDataSource control, 115, 404, 406
square brackets ([]), in SELECT statement, 122
Start option, Debug menu, 302
Start phase, of page life cycle, 265
state, 261, 267–269
  (see also application state; control state; session state; view state)
state bag, 275–279
state, restoring for AJAX controls, 77–83
StateBag application, 275–279
stateless environment, Web as, 267
Step Into option, Debug menu, 302
Step Out option, Debug menu, 303
Step Over option, Debug menu, 302
Stop Debugging option, Debug menu, 302
stored procedures, 146
StringCollection class, 360
strings, Visual Basic, 55
strong passwords, 330
strongly typed properties, LINQ object model, 148
Structured Query Language (see SQL)
Studio 8 (Total Training course), 39
style attribute, <input> element, 208
style classes, 213
<style> element, 210

style sheets, 207, 208, 212–220
  attaching to a page, 216–218, 392
  colors in, specifying, 216
  compared to master pages, 220
  creating, 85, 213–218, 389–392
  units of measurement used in, 216
styles, 207
  applying to elements, 208
  document level styles, 208, 210
  external styles, 208
  inline styles, 208–210
  for selectors, 213
stylesheet themes, 371, 377
StyleSheetTheme attribute, Page directive, 371
StyleSheetTheme property, Page, 377
subs, methods implemented as, 30
Substring( ) function, 137
Summary table, 52–54
SuppressPostBack property, CollapsiblePanelExtender control, 100
symbolic debugging, 291, 300
  breakpoints for (see breakpoints)
  configuration for, 9, 301
  Debug toolbar for, 302
  evaluating expressions during, 311
  examining methods and functions on call stack during, 313
  examining variables and objects during, 310–313
  stepping out of methods or functions, 310
  stepping over specific methods or functions, 310
  stepping through code, 309
  testing web site using, 9–11
  windows available during, 310–313
synchronous postbacks, 23, 27–29
syntax errors, 314
system requirements, 453

T

Table control, 42
tables, database, 113, 114
  binding to GridView control, 113–121
  schema names for, 121–123
tables, HTML
  creating, 39–40
  for page layout, 42, 392
  Summary table, 52–54
Target property, HyperLink control, 57
TargetControlID property
  CollapsiblePanelExtender control, 100
  PopupControlExtender control, 91
  TextBoxWatermarkExtender control, 87, 91
templates for data controls, 112
templates for web site
  choosing, 3
  with Dynamic Data, 162
Text property
  Button control, 24, 32
  Label control, 11
  ListItem object, 46, 78
  TextBox control, 41, 140
text, displaying, 52–54
TextAlign property, Image control, 56
TextBox control, 40, 41, 208
TextBoxWaterMarkExtender control, 84–89, 91–94
TextMode property, TextBox control, 41
Theme attribute, Page directive, 377
themes, 371
  creating, 374
  customization themes, 371, 379
  enabling, 375
  location of, 374
  setting for page, 377–380
  stylesheet themes, 371, 377
Themes application, 372
this keyword, C#, 237
tool tips, 45
Toolbox, 7, 8
ToString() method, 414
Total Training courses, 39
Trace attribute, Page directive, 296
trace log, 296–300
Trace object, 298–300
tracing, 291, 295–300
Transact-SQL Programming (Kline et al.), 412
TreeView control, 246–248
try/catch blocks, 291
type conversion, 139
Type property
  CompareValidator control, 193
  RangeValidator control, 196

U

unique key, 114
units of measurement in style sheets, 216
Universal Resource Locater (URL), 261
Unload event, Page, 267

UPDATE statement, generating for DataSource, 128–133
UpdateCommand control, 133
UpdatePanel control, 30–34, 71–76, 414, 425
  GridView in, 125
  in PopupControlExtender, 89, 92
URL (Universal Resource Locater), 261
user controls, 20
user input, verifying (see validation of data)
users
  anonymous (see anonymous personalization)
  authentication of (see authentication)
  creating, 327–330, 332–336, 396–397
  customizing web site based on (see personalization)
  logging in (see logging in)
  password requirements for, 330
  roles for (see roles)

V

Validate method, 265
ValidateEmptyText property, CustomValidator control, 199
validation controls, 20, 180
Validation Expression property, RegularExpressionValidator control, 197
validation of data, 179–182
validation of users (see authentication)
Validation phase, of page life cycle, 265, 267
ValidationGroup property, validation controls, 181
ValidationSummary control, 188, 420
Value property, ListItem object, 46, 78
ValueToCompare property, CompareValidator control, 192
variables, 55
  examining during debugging, 310, 311, 312, 313
  naming conventions for, 26
.vb files (Visual Basic code-behind files), 19, 153
view state, 268, 269–279
  disabling, 269, 270
  security of, 269, 270
  state bag used with, 275–279
  using, 270–275
  when not to use, 270
ViewState application, 270–275
ViewState keyword, 275–279
virtual directories, 466, 467–469
Visible property, Image control, 56
Visual Basic
  &=, concatenate assignment operator, 278
  -=, decrement assignment operator, 278
  /=, divide assignment operator, 278
  +=, increment assignment operator, 278
  _, line continuation character, 12
  *=, multiply assignment operator, 278
  access modifiers, 236
  arrays, 276
  Boolean variables, 28
  code-behind files (.vb files), 19, 153
  compared to C#, xiii
  conditional operators, 31
  CType method, 139
  dictionaries, 276
  errors, catching, 252
  For Each loop, 251
  For loop, 295
  If-Then statements, 31
  Me keyword, 237
  methods, implementation of, 30
  Mod operator, 200
  namespaces, 25
  object-oriented terminology for, 21
  Select Case statement, 283
  strings, 55
  use of, 4
  variables, 55
Visual Studio 2008, installing, 461–463
VWD (Visual Web Developer), installing, 454–457

W

Warn method, Trace object, 298–300
WAT (Web Site Administrative Tool)
  roles, creating, 340, 396
  users, creating, 327–332, 396
Watch window, for debugging, 311, 313
WatermarkCssClass property, TextBoxWatermarkExtender control, 87
watermarks, 85–89
WatermarkText property, TextBoxWatermarkExtender control, 87, 91
web pages (see pages)
web server
  processing of pages by, 18
  used by IDE, 5

Web Site Administration Tool (see WAT)
web site applications, 17
  appearance (design) of, 39
  building (compiling), 376
  copying
    with IDE, 471–475
    without IDE, 471
  creating, 2–5, 469
  developing remotely, 5
  directory for, 466–471
  event-driven, 11
  publishing (see publishing web sites)
  testing while developing, 9–11
web site resources
  AdventureWorks database, 458
  AJAX Control Toolkit, 464
  application examples, 388
  code examples
    in C#, xiii
    permission to use, xv
  DiscountASP.NET, 481
  for this book, xiii, xv
  HTTP status codes, 296
  Hungarian notation, 26
  Jesse Liberty, xv
  Network Solutions, 479
  Silverlight, 34
  strong passwords, 330
  Visual Studio 2008, 461
  Visual Web Developer (VWD), 454, 457
  web site design, 39
web.config file (see configuration file)
WebSiteNavigation application, 241–243
Welcome page, 336
well-known ports, 343
Wenz, Christian (Essential Silverlight), 34
When Breakpoint Is Hit dialog box, 308
WHERE clause, 142
Width property, Label control, 24, 30
Williams, Robin (The Non-Designer's Design Book), 39
Windows authentication, 118, 327
windows, manipulating, 6, 8
Write method, Trace object, 298–300

X

XHTML, AJAX using, 70
Xie, Donald (Programming C# 3.0, Fifth Edition), 147
XML, AJAX using, 70
XSLT, AJAX using, 70

Z

The Zen of CSS Design: Visual Enlightenment for the Web (Shea; Holzschlag), 39

About the Authors

Jesse Liberty is the best-selling author of O'Reilly's *Learning ASP.NET with AJAX*, *Programming C#*, *Programming .NET 3.5*, and a dozen other books on programming. He is a senior program manager at Microsoft on the Silverlight Development Team, and a frequent contributor to O'Reilly Network web sites and publications. Jesse is a former distinguished software engineer at AT&T and vice president for technology development at CitiBank, and was an independent consultant for 12 years.

Dan Hurwitz is the president of Sterling Solutions, Inc., where for more than two decades he has been providing contract programming and database development to a wide variety of clients.

Brian MacDonald is a technical editor specializing in Microsoft .NET programming topics. He has edited *Programming C#*, *Programming ASP.NET*, and *Programming WCF* (all from O'Reilly). He is also the coauthor of *Learning C# 2005*, *Learning C# 3.0*, and *Learning ASP.NET 2.0 with AJAX*. He lives in southeastern Pennsylvania with his wife and son.

Colophon

The image on the cover of *Learning ASP.NET 3.5*, Second Edition, is a monkfish (*Lophius piscatorius*). Europeans occasionally call the monkfish a sea monk, a kinder term for a fish also known as a sea monster.

Although according to Seafood Watch, Americans concerned with sustainable living practices should avoid eating the fish, the monkfish remains a popular dish among consumers and chefs alike.

Chefs have come to appreciate the fish—often labeled the "poor man's lobster"—for its ability to absorb the flavor of accompanying broths and spices. While the fish is only similar to the lobster in texture and not flavor, many chefs find it easier to promote the fish under a name the public will recognize and not find intimidating.

Its large head, which is twice as wide as its tail, and enormous mouth filled with sharp, pointy teeth give the monkfish a reputation for being one of the ugliest fish around. Markets will usually sell the monkfish without its head, a precedent perhaps set by France, where it had previously been illegal to bring monkfish into the country with their heads attached, as the fish might frighten passersby.

The cover image is from *Dover's Animals*. The cover font is Adobe ITC Garamond. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSansMonoCondensed.